AD-A100 822   AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN OF A LOCAL COMPUTER NETWORK FOR THE AIR FORCE INSTITUTE --ETC(U)
MAR 81 W C HOBART
UNCLASSIFIED AFIT/GE/EE/81M-3   NL

1 OF 4
AD A
100822

DDC

1
BS

LEVEL II

ADA100822

DTIC FILE COPY

DTIC
ELECTE
JUL 1 1981
S D
D

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

81 6 30 040

AFIT/GE/EE/81M-3

Accession For
NTIS GRA&I ☒
DTIC TAB ☐
Unannounced ☐
Justification

By
Distribution/
Availability Codes
Dist | Avail and/or Special
A

DESIGN OF A LOCAL COMPUTER NETWORK FOR THE AIR FORCE INSTITUTE OF TECHNOLOGY DIGITAL ENGINEERING LABORATORY

THESIS

AFIT/GE/EE/81M-3 William C. Hobart, Jr.
Captain USAF

Approved for public release; distribution unlimited.

DTIC

AFIT/GE/EE/81M-3

DESIGN OF A LOCAL COMPUTER NETWORK FOR THE AIR FORCE INSTITUTE OF TECHNOLOGY DIGITAL ENGINEERING LABORATORY

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

William C. Hobart, Jr., B.S.
Captain USAF
Graduate Electrical Engineering

March 1981

Approved for public release; distribution unlimited.

Preface

This work presents a design of a local computer network for the Digital Engineering Laboratory which I hope will provide a sound basis for the follow-on implementation efforts and later network expansion. The design is based upon techniques developed by Tom DeMarco, Edward Yourdon, and Larry Constantine and I gratefully acknowledge the part their techniques played in making my task simpler.

I would like to express my deep appreciation to Dr. Gary B. Lamont, who as my research advisor gave me valuable guidance and encouragement. Also, I thank my thesis readers, Dr. Thomas Hartrum and Captain Walter Seward whose constructive comments helped to improve the clarity of this thesis. In addition, I am indebted to the faculty members whom I interviewed to assess the requirements of the Digital Engineering Laboratory for a local computer network.

Finally, I wish to thank my wife, Linnea, for her help and encouragement during the past year.

William C. Hobart, Jr.

Contents

List of Figures

Figure Page

List of Tables

Abstract

A local computer for the Air Force Institute of Technology Digital Engineering Laboratory was designed and the network command language interpreter modules were implemented. The requirements for this network were specified by interviewing nine faculty members associated with the Digital Engineering Laboratory and then translating their functional requirements into a detailed set of hardware and software system requirements. Structured Analysis was used to produce a structured specification for the applications, host-to-host, network, and link protocol requirements. Yourdon and Constantine's Transform Analysis and Transaction Analysis techniques were then used to develop a set of module structure charts for the software design. The network uses a loop topology for the nodes with a star of up to four hosts connected to each node. The nodes are implemented using a Universal Network Interface Device (UNID) developed at the Air Force Institute of Technology. Initially, the network will include an Intel Series II Microcomputer Development Station, a Digital Equipment Corporation VAX-11/780, and a Data General Nova. These computers will be connected to the nodes using twisted pair and the two UNIDs in the initial configuration will be interconnected with a duplex fiber optic link supporting transmission rates up to 56 Kbs. The X.25 protocol was selected to implement a host-to-host transfer mechanism in conjunction with a basic routing algorithm using a lookup

table stored in each UNID. The network command language interpreter allows file transfer commands, session control commands, and user help requests to be parsed and the appropriate parameters passed to lower-level modules.

I. Introduction

The purpose of this investigation was to design a local computer network for the Air Force Institute of Technology School of Engineering Digital Engineering Laboratory. This network, named the Digital Engineering Laboratory Network (DELNET), was first proposed in 1978 when the number of host computers in the Digital Engineering Laboratory had grown to six and there were insufficient peripheral devices to allow the computers to be used to their full capacity. Local networks at this time were becoming increasingly commonplace in universities and corporations. This was due to their desire to increase their information processing power without adding additional computers. Also, universities wishing to perform research in the area of local computer networking were developing their own networks so that they could have the flexibility to change network characteristics and evaluate different network topologies, protocols, and transmission mediums. Interest in both of these potential activities provided the impetus for the development of a local computer network at the Air Force Institute of Technology.

Historical Perspective

It is only in the last decade that computer networks have become commonplace. The development of the Defense Advanced Research Projects Agency Network (ARPANET) probably did more to stimulate the development of computer networks than any other factor. This packet-switching network has been highly successful and today continues to be the primary computer network in the United States and ties together the most powerful research computers in the country (Ref. 12: 5-23).

It was not until the advent of the microprocessor, however, that local networks tailored to a particular organization and interconnecting minicomputers became practical. The microprocessor made local networks more practical because it allowed economic network nodes to be developed. These were cost-effective and yet offloaded much of the network protocol processing overhead from the computer hosts on the network. Also, the microprocessor decreased the cost of computers and increased the number in use. This made it practical for organizations to implement a local network of microcomputers in lieu of time-sharing off a mainframe computer.

Another significant development has been the emergence of various link and network protocols. These protocols have started to be standardized, but because protocol design is still in its infancy, there is no recognized "best" protocol for a given application (Ref. 20: 156). IBM has played a leading role in protocol development first by introducing the binary synchronous (BSC) protocol and later by introducing the Systems Network Architecture (SNA) which includes the synchronous data link control (SDLC) protocol. SDLC was one of the first bit-oriented protocols; a set of protocols which has emerged as the most efficient and flexible for managing the link between two computers. The X.25 protocol, which defines the interface of a computer to a packet-switching network, has just recently become a standard and could be the initial protocol for interfacing computers in the next decade (Ref. 3: 11).

A great deal of research has been conducted to design efficient routing algorithms which direct the flow of the packets through the switching nodes in a network. Although a number of efficient algorithms have been developed, this area of research is also still in its early stages (Ref. 16: 42-97).

Until very recently, most local computer networks available commercially were developed by a computer firm to interface that manufacturer's systems. The only way of interfacing other computers to that network was to develop an emulator. This emulator would make that computer appear at the interface as one of the manufacturer's computers to the network. IBM's SNA and the Digital Equipment Corporation Network (DECNET) are both examples of this type of network (Refs. 3: 11; 7). In the last two years, two new local networks have become commercially available that can interface a heterogeneous set of computers. Xerox has introduced ETHERNET and Ungermann-Bass has developed NET/ONE (Refs. 5,14). Because of the state of the art techniques used in these networks, they are worth studying in more detail.

NET/ONE uses a network interface unit (NIU), which may contain up to four Z-80A microprocessors and 64 Kbytes of memory, to interface up to sixteen devices to the network. These devices may include host computers or peripheral devices and are interfaced to the NIU using the RS-232 protocol. NET/ONE uses a bus architecture and interfaces the NIUs to the bus through a transceiver interface contained in each NIU. The bus is a baseband coaxial cable supporting transmission speeds of up to 4 Mbs. Because the NIU is connected to this cable through a passive tap, a failure of an NIU only affects those hosts and devices on it. The bus access mechanism uses a contention channel protocol where each NIU can transmit whenever it senses that the bus is idle. If it receives another NIU's transmission while transmitting data itself, then a collision has occurred and the packet is retransmitted a set delay after the NIU again senses that the bus is idle (Ref. 5).

ETHERNET is very similar to a NET/ONE without the NIUs. Instead each host must interface to a coaxial cable through its own transceiver interface and handle the contention channel protocol itself (Ref. 14).

Thus computer networks have evolved from a few geographically distributed networks interconnecting large mainframe computers such as ARPANET to also include high speed local networks such as NET/ONE and ETHERNET. These two local networks make the interconnection of a heterogeneous set of minicomputers and microcomputers practical. The standard protocols and routing algorithms developed to date are the principal resources from which the design of DELNET could be derived. Existing local networks such as ETHERNET and NET/ONE also provide a basis for comparison with the DELNET design. So, it is from this perspective that the development of DELNET has taken place. The following section gives some background on the previous development efforts which preceded this particular investigation.

Background

This effort follows two other graduate research projects addressing the development of a local computer network for the Digital Engineering Laboratory. R. Cade Adams and Donald Ravenscroft conducted concurrent investigations although with different emphases (Refs. 1,16).

Adams' investigation resulted in several general theoretical concepts upon which that he felt that the design of DELNET should be based. However, the conceptual design was based upon only a cursory analysis of the actual Digital Engineering Laboratory requirements and more upon requirements of local networks in general (Ref. 1).

Ravenscroft's investigation was much more productive and resulted in the design of a routing algorithm for DELNET. Also, some recommendations were made for the choice of a link protocol (Ref. 16). However, neither of these investigations approached the problem from the users' viewpoint and much work remained to develop a design for DELNET that could be implemented in the Digital Engineering Laboratory.

Objective of This Investigation

The objective of this investigation was to specify the design of DELNET in sufficient detail that subsequent investigations could concentrate on actually implementing the design. The design had to be based upon the actual requirements of the Digital Engineering Laboratory and their projected uses of DELNET. This was imperative because unlike ETHERNET or NET/ONE that were designed for high throughput and automated data processing, one of the chief uses of DELNET will be as a teaching and research tool. Thus, flexibility in changing the protocols and topology of the network were important for DELNET and these requirements could not be met by NET/ONE or ETHERNET because of their dependence on the bus architecture and the contention channel protocol.

Approach

The development of computer networks has not reached a stage where there is a set development approach to be followed. However, there are standard procedures for the engineering development of any system and it is from these procedures that an approach was formulated.

First, a top-down development of the design was chosen. This approach allowed the design to first address those levels closest to the user and thereby insure that the design was consistent with the user's requirements. Then lower levels of the design were developed to support the requirements of the next higher level. Care was taken, however, to insure that each level had sharply defined interfaces with the levels above and below it so that a level designed with one protocol could be replaced easily with another protocol when desired.

Due to the size of the development effort, an approach using structured analysis and design techniques was considered imperative. These techniques result in clear written specifications and diagrams that provide to those implementing the design the unambiguous information they need.

The first phase of the investigation consisted primarily of researching the literature and gaining a working knowledge of computer networks, protocols, hardware interface devices, and the capabilities of the computers in the Digital Engineering Laboratory. Existing local computer network architectures were studied in particular as were the newer bit-oriented protocols being used increasingly in local computer networks.

Once a sufficient background had been gained, a user interview outline was designed and faculty members associated with the Digital Engineering Laboratory were interviewed. From these interviews, a set of projected uses and functional requirements were compiled and these were used to derive the system requirements.

The system requirements consisted of both hardware requirements for the topology, hosts, nodes, and transmission mediums, as well as software requirements. The software requirements were documented using Structured Analysis (Ref. 6). The structured specification consisted of a set of data flow diagrams and a data dictionary.

The system requirements were then used to develop a hardware design and a software design of the high-level protocols. These protocols were then partially implemented on the VAX-11/780 and user feedback on the performance was obtained. The final stage of the investigation resulted in the design of the lower-level protocols.

Overview of the Thesis

The structure of the thesis basically follows the approach that was taken in the investigation. The DELNET functional requirements are analyzed in Chapter II. Appendix A provides supporting information including the compilation of the results of the user interviews and a list of those interviewed and their specific areas of interest with respect to the Digital Engineering Laboratory.

Chapter III translates the functional requirements into hardware and software requirements. Structured Analysis is described and data flow diagrams are used throughout the chapter to support the written description of the software requirements. Supporting the hardware requirements is Appendix B which contains a floor layout of the location of the principal computers in the Digital Engineering Laboratory. Appendix C contains the complete structured specification in support of the software requirements.

The design phases of DELNET are described in Chapter IV. The design of the hardware is specified including the topology to be employed initially, the first hosts to be included in the network, the node to be used in the network, and the transmission mediums to be used. The software design is described in three stages. First, the design techniques that were used are summarized and module structure charts from the DELNET design are used to illustrate their application. Then, the allocation of the software modules to the host and node processors is specified. The actual software design consists of a set of module structure charts which are in Appendix D.

Chapter V describes the implementation and testing of the network command language interpreter. In this chapter, the factors affecting implementation are described as well as the testing techniques employed. The source code for the implemented modules is in Appendix E while Appendix F contains the testing documentaion. Appendix G contains the user manual for DELNET.

Finally, Chapter VI summarizes this investigation and gives recommendations for follow-on research efforts.

II. DELNET Functional Requirements

Introduction

This chapter specifies the functional requirements of DELNET. First, the background of the requirements analysis is discussed. In the next section, the content of the user survey that was used to help determine the DELNET requirements is described. Also included in this section is a description of how the survey was conducted. The next section summarizes the results of the survey by describing the projected uses of DELNET as well as the users' perception of several functional requirements. Finally, other functional requirements as well as constraints not addressed by the user survey are discussed.

Background

Possibly the most crucial step in developing a computer network is the specification of the network requirements. Yet, this phase of development was treated inadequately by both individuals whose work this research project was continuing (Refs. 1,16). While Adam's thesis discusses user requirements, it only lists characteristics generally desired of local networks and does not justify these characteristics as being applicable to DELNET. Also, there is no reason to believe that this list of requirements is complete (Ref. 1). In Ravenscroft's thesis, some attention is given to the pedagogical requirement for DELNET as well as many of those requirements addressed by Adams, but Ravenscroft's requirements analysis is also incomplete (Ref. 16: 43). Thus, many of the design decisions that follow in both theses are only based upon what is generally desired in networks.

While it is possible to specify many requirements that are generally desired in a network, using this approach alone will result in an incomplete and inaccurate requirements specification for several reasons. First, local networks differ greatly in their requirements because they usually serve only one organization (Ref. 21: 131). Thus, the requirements for local networks are usually as varied as the organizations themselves. Second, not all requirements for a network may have the same importance and the resources available (such as time or money) may not be adequate to satisfy all requirements. Since, the relative importance of the requirements also varies widely with each local network, it is almost impossible to obtain a weighting of requirements by strictly considering some "ideal network" for all cases. Finally, there may be unique requirements for the network and these will be overlooked if only the requirements for the "ideal network" are considered. These pitfalls were demonstrated when the two previous research efforts, which were concurrent efforts, resulted in conflicting designs. Ravenscroft did not consider the topology recommended by Adams because of its inflexibility, while Adams rejected the topology recommended by Ravenscroft because of its complexity.

User Survey

A systematic approach was needed to specify the requirements for DELNET that would tailor it to the requirements of the Digital Engineering Laboratory. Thus, a three-part user survey was designed.

In the first section, the users were asked what applications could be served by DELNET from their utilization perspectives. To aid the users being interviewed, nine common local network applications were listed which the user could evaluate on a five-point scale from "of no use" to "very beneficial." These applications included peripheral sharing among the computers in the network, file accessing and transfers across the network, sharing software tools on the network, accessing the Advanced Research Projects Agency Network (ARPANET), accessing the CYBER 750 host installation computer, accessing the Air Force Institute of Technology Network (AFITNET) once it is implemented, doing distributed processing, managing distributed databases, and providing fault tolerance. Finally, the section asked the users to identify the applications that they felt should be implemented first.

The second section asked the user to estimate the requirements from their perspectives for nine basic network parameters. These were divided into five user-oriented parameters and four design-oriented parameters. The user-oriented parameters were throughput, response time, the user interface, security, and availability of the network. Each of the subsections addressing a parameter had a number of questions to aid the user in evaluating the requirement for that parameter. Each user was asked to identify any other user-oriented parameters that might influence DELNET's design. The design-oriented parameters were flexibility, performance monitoring, special pedagogical requirements, and the availability of a distributed processing language. As with the user-oriented parameters, each subsection had a number of questions to help the user evaluate the parameter. The user was also asked to identify other design-oriented parameters. Finally, the second section concluded by having the user rate each of the nine parameters on a five-point scale from "not applicable" to "essential." This was included to help evaluate the relative importance of the network parameters.

The third section asked the users to make any other comments that they felt might help specify the requirements of DELNET.

The identification of the users of DELNET was also an important part of the requirements analysis. Although undergraduate and graduate students will use DELNET, their use of DELNET would be primarily at the direction of the Electrical Engineering (EE) Department and the Mathematics Department. Thus, the students had little interest or insight into the projected uses of DELNET. On the other hand, many faculty members had expressed an interest in DELNET and knew what capabilities would be most useful to the EE department. Because of this, seven EE faculty members and two mathematics faculty members were chosen to be interviewed. Each of those chosen represented a specific area of interest and so the results of the interviews represented a wide range of user applications.

Each user was interviewed for approximately forty-five minutes to an hour and a half using the user survey described above. Personal interviews were chosen over regular surveys to allow each user to ask for clarification of the various questions, and also in hope of getting more information from the users than they might have given in written responses to the survey. The response to the user interviews was excellent and the information from the user survey was used to formulate the general requirements specifications that follow.

Projected Uses of DELNET

The user survey results were used to determine the projected uses of DELNET. The principal uses that were considered most beneficial were in the area of resource sharing. Specifically, peripheral sharing and file access and transfer capability across DELNET were identified by all users as being of top priority. Next in potential benefits was the capability to access the CYBER 750 from DELNET. Access to AFITNET, another local computer network, being developed to process much of AFIT's ever-increasing workload, was regarded as equally beneficial once this network is implemented. Software tool sharing was also identified as beneficial while access to the ARPANET was considered of lesser benefit. Distributed processing as well as distributed databases received widely varying evaluations. Those with a special interest in these areas rated these capabilities as very beneficial while others rated them of little use. There was a general concensus that the primary benefits of these capabilities would be pedagogical although one person felt that the importance of distributed processing would increase rapidly in the next few years. Because the network was not expected to support critical functions, no need was perceived for the network to guarantee uninterrupted service to the user through redundancy and fault-tolerance. Fail-soft capability was regarded as beneficial, however. The projected uses of DELNET are summarized in Table 1.

User-oriented Functional Requirements

The users' functional requirements were also clarified by the survey. The throughput required by the users was basically driven by the need to be able to transmit 16-32 Kbyte files between the computers on the network in less than three minutes. Usage of the computers in the Digital Engineering Laboratory should average about 3 to 4 hours per day with peaks of up to almost 24 hours a day for a few days at a time on some computers like the Data General Eclipse. The computers with the heaviest projected usage in the

Table 1

Projected Uses of DELNET

Projected Use	Very Beneficial	Beneficial	Somewhat Beneficial	Of Little Use	Of No Use
Peripheral Sharing	7	0	0	0	0
File Transfers	5	2	0	0	0
Software Tool Sharing	2	4	1	0	0
Access to CYBER 750	3	3	1	0	0
Access to AFITNET	3	4	0	0	0
Distributed Processing	2	1	2	2	0
Distributed Databases	2	0	2	3	0
Fault Tolerance	0	1	1	3	2

Digital Engineering Laboratory are the Data General Nova, the Data General Eclipse, the Digital Equipment Corporation VAX-11/780, the Digital Equipment Corporation LSI-11s, the Intel Series II MDS, the Digital Equipment Corporation PDP-11s, and the Microprogrammable Minicomputer Emulator (MIME) that was built in the Digital Engineering Laboratory.

The response time requirement was, of course, closely tied to the throughput requirement. The requirement for response times was addressed for three modes: interactive, file transfers, and echoing user inputs. In the interactive mode, two to three seconds for "simple" commands was considered to be satisfactory. For file transfers, ten to fifteen seconds was given by one user, although this is not feasible on many of the Digital Engineering Laboratory computers even when the user is in the local computer mode. Thus, the file transfer response time was set at three minutes for a 32 Kbyte file. This was considered an acceptable tradeoff between user requirements and the file accessing capabilities of the computers in the Digital Engineering Laboratory. The echo response time requirement was given as being a maximum of one-half second. This is to insure that the echoes of the user inputs do not interfere with their entering subsequent data at the keyboard. With the exception of the ten to fifteen second file transfer requirement, all other response time requirements seemed consistent with the results of psychological studies predicting satisfactory levels of performance for the typical user (Ref. 13: 322,323). Another requirement on the response time was that the standard deviation for the distribution of response times be less than half the mean response time to the command. Thus, consistency in the response times was also considered important.

In addressing the requirements of the user interface, the key requirement that was mentioned by almost all users was the need to make the network configuration and specific operating systems of the hosts transparent to the user. The difficulty of achieving this was also mentioned, however, because of the difficulty in concealing operating system idiosyncrasies. Other user interface requirements included error recovery capabilities and the capability for the user to get help from the network through a "teach" command. One user also felt that the transparency should be optional so that the user can access capabilities of the machines that the network command language may not exploit.

Security was addressed from three perspectives. All users indicated that they did not forsee the possibility of running classified data on any of the computers in DEL. Thus, the concensus was that the network did not need to be designed to safeguard the processing of classified information. The second aspect of security addressed the requirement to protect files on the network from unauthorized access or alteration. Two-thirds of the users felt that this was a requirement while one-third saw no need for this capability. One user also specified that this access control should allow users to be given various levels of access rights such as "read only". Thus, file access restriction was found to be a security requirement for the network. The third aspect of security was access control to the network from outside DEL. Since gateways to the CYBER, AFITNET, and possibly the ARPANET were envisioned, the need for access control at these gateways was evident.

Defining the availability of the network was the first problem encountered in trying to assess the availability requirement. The availability of DELNET was defined to be the percentage of time that the network provided the capabilities required by a particular user as compared to the time that it was supposed to provide those capabilities. This definition allowed the users to individually assess their requirements for network availability. The assessments ranged from 60 percent to nearly 100 percent with 90 percent being the answer given by almost half of the users. The time periods that the network should be available were identified as either during the duty hours of the Digital Engineering Laboratory technicians or during the hours that the CYBER 750 is available. The lower percentages were given by those who stated what they felt would meet their needs, while the highest percentage was given because the user felt that this should be fairly easy to achieve. The 90 percent availability level met all stated requirements of the users who were interviewed and thus represents a reasonable target goal for the system design. The user-oriented functional requirements are summarized in Table 2.

Design-oriented Functional Requirements

The design-oriented requirements included flexibility, performance monitoring, and pedagogical requirements. The flexibility requirement was addressed by asking the users to describe how they would see the network changing over the next five years. The response given by almost all was that more hosts and devices would be added to the network. All

Table 2

User-oriented Functional Requirements

Area	Not Applicable	Marginally Applicable	Applicable	Very Applicable	Essential
Throughput	1	1	3	2	0
Response Time	0	1	2	3	1
User Interface	0	0	1	3	3
Security	2	1	1	1	2
Availability	0	1	1	3	2

the users felt that it was very important for DELNET to be easily reconfigurable with respect to adding new hosts and devices. Other changes mentioned included increased use of the network by those outside the Digital Engineering Laboratory and a transition of the network workload from predominantly file transfers to more interactive bursty traffic. Flexibility with respect to the topology, protocols, and transmission medium were generally considered requirements only from the pedagogical point of view. One user also felt that it was important to be able to vary between serial and parallel bit transmission.

The need for some form of performance monitoring capability was expressed by all users. They felt that statistics of the traffic through each node as well as accounting data collection on user jobs were both important to monitor. Both hardware and software monitor capabilities were stated as requirements by two of the users and another user suggested incorporating a performance monitoring node into the network. Thus, although the capability to monitor the performance of DELNET was considered important by all users, the requirements for implementing this capability varied significantly.

The pedagogical functional requirements given by the users are basically reflected in the requirements above for flexibility and performance monitoring. Another pedagogical requirement that was expressed was that DELNET should have the capability to evaluate the use of fiber optic links in computer networking.

Given the possibility of implementing a distributed processing capability, the requirements for a distributed processing language that would be available on all machines were assessed. Pascal was mentioned by all users (one specified UCSD Pascal) and Ada and FORTRAN were mentioned by almost all. BASIC was mentioned by one user and one user stressed the capabilities that Ada would have in distributed processing due to its powerful control structures.

The users were also asked to rank each of the functional requirement areas on a scale from not applicable to essential. Flexibility received the highest overall rating falling between very applicable and essential. The user interface, the availability, performance monitoring capability, and the pedagogical requirements were all rated as very applicable. The response time requirements fell a little lower between applicable and very applicable, while the throughput requirement, security, and the requirement for a distributed processing language were only considered as applicable. The design-oriented functional requirements

Table 3

Design-oriented Functional Requirements

Area	Not Applicable	Marginally Applicable	Applicable	Very Applicable	Essential
Flexibility	0	0	0	4	3
Performance Monitoring	0	1	0	4	2
Pedagogical	0	1	0	4	2
Distributed Processing Language	0	2	2	3	0

are summarized in Table 3.

Other User Requirements

The third section of the user interview produced eight additional requirements for the network. One requirement was that the network have a general interface port where student digital hardware design projects could be interfaced to the network. Another request was that the intercommunication between the hosts and the network be interrupt-driven. The ability to find a file in the network that had been sent from one host to another was also mentioned as a requirement. One user felt that there was a very great danger of letting the potential capabilities of DELNET determine how it would be used rather than addressing how DELNET could best fulfill the needs of the Digital Engineering Laboratory. In particular, he felt that the DELNET should not be used for general software development but rather that this should be done on the CYBER 750. It was pointed out by another user that the requirement to have all hosts available on the network is probably only applicable about one percent of the time when computer networking research is being accomplished. Another user felt that DELNET should be able to be easily initialized to contain an arbitrary subset of the hosts available in the network. The number of terminals and their locations was another requirement that a user felt needed to be addressed. Another user felt that the network should serve as a testbed for the Universal Network Interface Device (UNID) that is also being developed here at AFIT and should be ready for testing shortly (Refs. 4,18). Finally, one user stated that top-down decomposition and structured analysis should be used in the design process.

Constraints on DELNET

Another requirement of DELNET was that its cost of implementation be less than $10,000 per year until 1983 when an additional $30,000 will be available for the network.

The noise environment was also assessed. Although ±100 volt variations in the voltage levels from the building power supply have been observed, the Digital Engineering Laboratory has its own power supply that can be used. Also, while noise contributions from all the computers and other electronic devices in the Digital Engineering Laboratory might be expected to be extensive, no problems attributable to a noisy environment have been encountered using twisted pair to link such computers as the LSI-11s.

The physical layout of the computers also had to be addressed. The computers that are candidates for the DELNET are all in the same building, on the same floor, and located in four adjacent rooms. The maximum separation of any two computers is about 125 feet. A floor plan showing the locations of all minicomputers having semi-permanent locations in the Digital Engineering Laboratory is included in this paper as Appendix B.

Finally, the maintainability requirement of the network had to be determined. In order to minimize the difficulties involved in maintaining DELNET, all software must be thoroughly documented and the hardware should be designed for easy modular replacement. This will allow a faulty module to be quickly replaced with a spare while it is being repaired so that DELNET can continue to be available.

Summary

Through the results of the user survey it was possible to specify not only a comprehensive list of functional requirements but also to estimate their relative importance. The ability to transfer files across the network and to share peripherals attached to other hosts on DELNET were the two most important uses and those that the users wished to see implemented first. Flexibility was considered the most important functional requirement; and the user interface, availability, performance monitoring capability, and the pedagogical requirements were also considered very important. Finally, other constraints such as development funds available, the physical location of the hosts, and the noise environment were assessed. Appendix A contains a complete compilation of the user survey results.

III. DELNET System Requirements

Introduction

This chapter translates the general functional requirements addressed by the last chapter into more detailed system specifications. The first section of this chapter addresses the system hardware requirements while the second section specifies the system software requirements. The principal hardware system requirements to be addressed are the network topology, the host computers to be included in the network, the selection of a suitable node for the network, and a choice of appropriate transmission mediums for the links on DELNET. The system software requirements are specified using DeMarco's Structured Analysis technique (Ref. 6). A very abbreviated description of the components of the technique is given and its use is justified. Then, Structured Analysis is used to specify the system software requirements at the user level, the applications level, the host-to-host level, the network protocol level, and the link protocol level. Finally, the physical protocol specifications are stated.

The requirements specification was actually an iterative process. While some of the specifications found in this chapter could be derived directly from the functional requirements, many actually followed from design decisions made in the following chapter. These requirements would then trigger new design decisions which might in turn modify the requirements specification further. What follows, then, is the final results of this iterative process.

System Hardware Requirements

The functional requirements of Chapter 2 can be used to derive rather detailed specifications of the hardware that is required. This is done in the following sections by considering how the functional requirements of the system place constraints on the system hardware. These constraints are then used to derive the more detailed hardware specifications.

Topology. The primary requirement for the topology was that identified by Ravenscroft and mentioned earlier--that the topology had to be flexible, and must lend itself to easy expansion through the addition of more hosts and nodes. However, the topology also must not have bottlenecks that will limit the throughput on the network below the required level or that will increase the response time to unacceptably high levels due to queueing delays. The response time requirement may also impact the topology by limiting the number of nodes between any two host computers since the combined queueing delays may increase the response time to an unacceptable level. Since availability was not a major concern, the topology need not be overly redundant.

Host Computers. The requirements for the host computers to be included in DELNET principally reflected their usefulness from a pedagogical viewpoint, their capabilities, and their peripherals. DELNET should allow computers to be added to the network easily, regardless of their sophistication. So, there was a requirement for both a simple microcomputer and a sophisticated minicomputer to be included in DELNET to demonstrate DELNET's capability to interface with both. Also, since peripheral sharing was a major projected use, those computers with the most powerful peripherals should be included in DELNET. Finally, the usefulness of the network would be enhanced, if those computers used most often in the Digital Engineering Laboratory were included in DELNET.

Nodes. The requirements for the nodes in the network also were addressed. Because DELNET is to be as transparent to the users as is practical, it should not noticeably degrade the performance of the host computers when they are included in the network. Thus, the nodes must be capable of handling lower-level network protocols. The nodes should also be easily reconfigured to accomodate various topologies and also must be capable of meeting the network throughput requirements. Furthermore, the node should have the capability of being easily reprogrammed for different protocols and should be able to collect performance monitoring statistics. Finally, if possible, a compiler should be available for the CPU that the node is based on so that the protocol software may be written in a higher order language. This would reduce the magnitude of the implementation effort and enhance the maintainability of the software.

Transmission Medium. There are several system requirements for the transmission medium as well. First, it must support the data transmission rates necessary to meet the throughput and response time requirements. Second, it must provide reliable communications links to avoid degrading throughput and response times. Otherwise, if a high percentage of blocks of data required retransmission then both response time and throughput would suffer. The same would be true if forward error correction was used and a high degree of redundancy was required. On the other hand, there is no requirement that the transmission medium provide secure communications or provide high noise immunity. The transmission medium should be easily re-routed, however, to allow the topology to be easily reconfigured. Finally, at least one link in DELNET should make use of fiber-optic technology to satisfy the pedagogical requirement stated in the user survey.

Structured Specification of the Protocol Requirements

Introduction. While the user surveys provided an excellent tool for specifying the functional requirements for DELNET, and even some of the more hardware-related system requirements, a technique was needed to translate the DELNET functional requirements into system software requirements. Several techniques were available for this task including DeMarco's Structured Analysis (Ref. 6), SofTech's Structured Analysis and Design Technique (SADT)

(Ref. 19), IBM's hierarchical input-process-output (HIPO) diagrams (Ref. 15: 17,18), and various problem statement languages (Ref. 2). Structured analysis was chosen as the technique to be used to translate the functional requirements into the system software requirements due to several advantages that it offers. However, before these advantages can be understood, it is necessary to understand the basic components of the Structured Analysis technique.

Structured analysis is a technique using data flow diagrams (DFD's) and a data dictionary which uses Structured English, decision tables, and decision trees to describe the DFD's. The DFD's have the four components shown in Figure

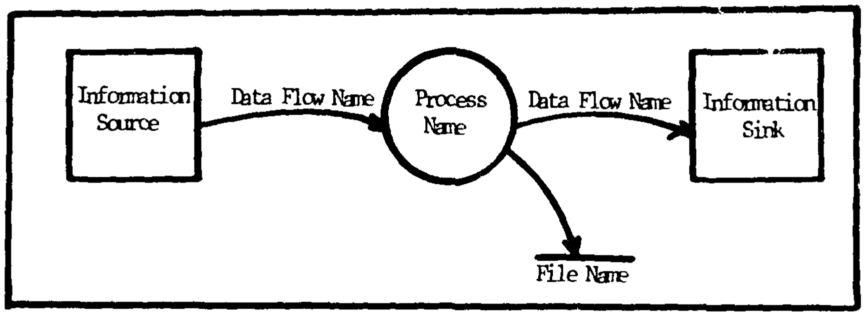


Figure 1 DFD Components

1. The first component is the data flow. It is a "pipeline" of other data flows and of data elements. The data elements are the basic data types that cannot be partitioned further and still retain their meaning. A data element might be a three-digit integer representing the length of a file in bytes or a seven-letter word representing a command type.

The data flow is represented by a curved arrow on the DFD's. The process converts input data flows to output data flows and is the second component of the DFD's. It is represented by a circle or bubble that contains the process name. The boxes represent the third component of DFD's, the sources and sinks of information. They may represent a user keyboard, a user display, or any other mechanism by which information enters or leaves the system. Finally, files are the last component and are repositories of information within the system. They are shown by straight lines. The DFD's are layered starting with a context diagram that defines the interface of the system with its environment. Then the processes in the diagram are expanded into lower-level DFD's. This partitioning process continues until the data flows entering and leaving a process consist of only one data element each. At this point, a process description is written for the process and the process is not expanded into a lower-level diagram. The process descriptions, data flow and data element descriptions, and file descriptions for all the DFD's are compiled into the data dictionary.

With the very abbreviated description of Structured Analysis given above, it is now possible to examine eight of the advantages that Structured Analysis offers. First, it was based upon the concept of partitioning. This allowed the complex network requirements to be addressed abstractly at the user level where the context diagram could represent the interface that the user wished to make to the system. Then lower level diagrams could be used to refine the data flows and processes in the higher level diagrams to sufficient detail. Thus, this method of partitioning allowed the large and complex DELNET requirements problem to be approached in an orderly manner rather than causing the analyst to be overwhelmed by the volume of requirements to be specified. Second, through the use of a data dictionary and the concentration on data flows rather than processes, Structured Analysis allowed the interfaces to be more clearly defined--in fact, it practically forced this to occur. Third, the process definitions could be easily specified using Structured English, a tool incorporated into structured analysis. Fourth, the structured specification consisting of the data flow diagrams and data dictionary was organized to eliminate redundancy in the structured specification. This made the task of maintaining the specification and changing it much easier. Fifth, the data flow diagrams were a two-dimensional presentation of the requirements and thus presented the structure of the DELNET requirements much clearer than the linear, voluminous presentation of a traditional specification. Sixth, Structured Analysis differentiated between the logical and the physical environment, thus more clearly defining the problem. This was also very helpful since this allowed the functional software requirements to be specified without being concerned about the actual hardware that would execute those requirements. Seventh, when a data flow diagram was wrong, it was usually glaringly wrong, and thus it was easier to spot inconsistencies and gaps in the structured specification and to correct the deficiencies. Finally, it was fairly easy to transition from the structured specification to the design phase since the data flow diagrams gave a first cut at the module structure charts (for Yourdon and Constantine's structured design technique) using Source/Transform/Sink analysis (Refs. 6,22). The major disadvantage was that it was very time consuming to generate the structured specification. However, the extra time spent in the requirements definition was probably more than recompensed by the time saved in the design phase and the benefits of having less requirements errors.

Context Diagram. The context data flow diagram shows the system boundary and interface with the user. As can be seen in Figure 2, the network operating system (NOS) must include everything between the user input from a keyboard that is connected to a computer in the network, to the response that the user receives at his display. Thus, the NOS includes all the computer operating systems in the network as well as all interface software and hardware required to implement the network functional requirements. To initialize the network, a configuration bootstrap process is also required. In addition, Table 4 shows the layers of protocol that are defined in the sets of DFDs that follow. Table 5 describes the process hierarchy for the first set of

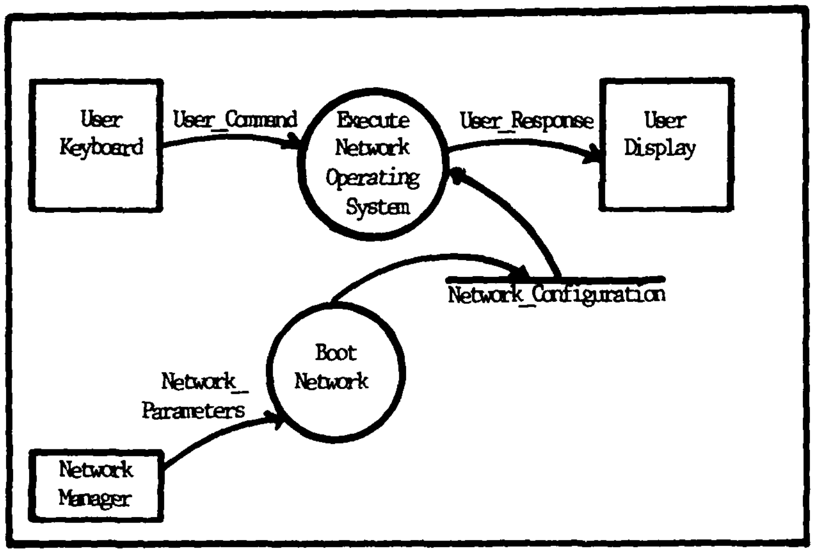


Figure 2 Context Diagram

Table 4

Layers of Protocol

Network Operating System
Host-to-Host Protocol (X.25 Level 3)
Network Protocol (Routing Algorithm)
Link Protocol (X.25 Level 2)
Physical Protocol (RS-232C)

DFDs, the Network Operating System.

System Diagram. The system diagram translates the key functional requirements into the software requirements for the NOS as shown in Figure 3. First, the user command must be examined to determine whether it is a network command or a local command (1). If it is a network command, then it must be sent to the computer with the software to interpret the network command (2). The transmitted network command must then be classified as either a command to transfer a

Table 5

Network Operating System Process Hierarchy

1. Determine Command Type

2. Send Command to Host with NOS

3. Determine Network Command Type

4. Transfer File
    - 4.1 Send Get-File Command to File Source Host
    - 4.2 Get File from Source Device
    - 4.3 Restructure File for Network
    - 4.4 Transmit File to Destination
        - 4.4.1 Divide File into Blocks
        - 4.4.2 Transmit Block
        - 4.4.3 Reassemble Blocks
    - 4.5 Restructure File for Dest Host
    - 4.6 Store File on Dest Device

5. Control Session
    - 5.1 Determine Session Control Command Type
    - 5.2 Log User Into Network
    - 5.3 Log User Out of Network

6. Help User
    - 6.1 Determine Help Info Requested
    - 6.2 Provide General Network Introduction
    - 6.3 Provide Procedure for Transferring Files
    - 6.4 Provide List of Active Hosts and Devices
    - 6.5 Provide Procedure for Logging Out of Network

7. Send Message to User's Host

8. Route Local Command

9. Execute Routed Local Command

10. Route Local Response

file from one device on the network to another device, a command to log into or log out of the network, or a request for help in using the network (3). First, if the request is to transfer a file, then the NOS must check the network configuration to insure that the command is valid. Then the

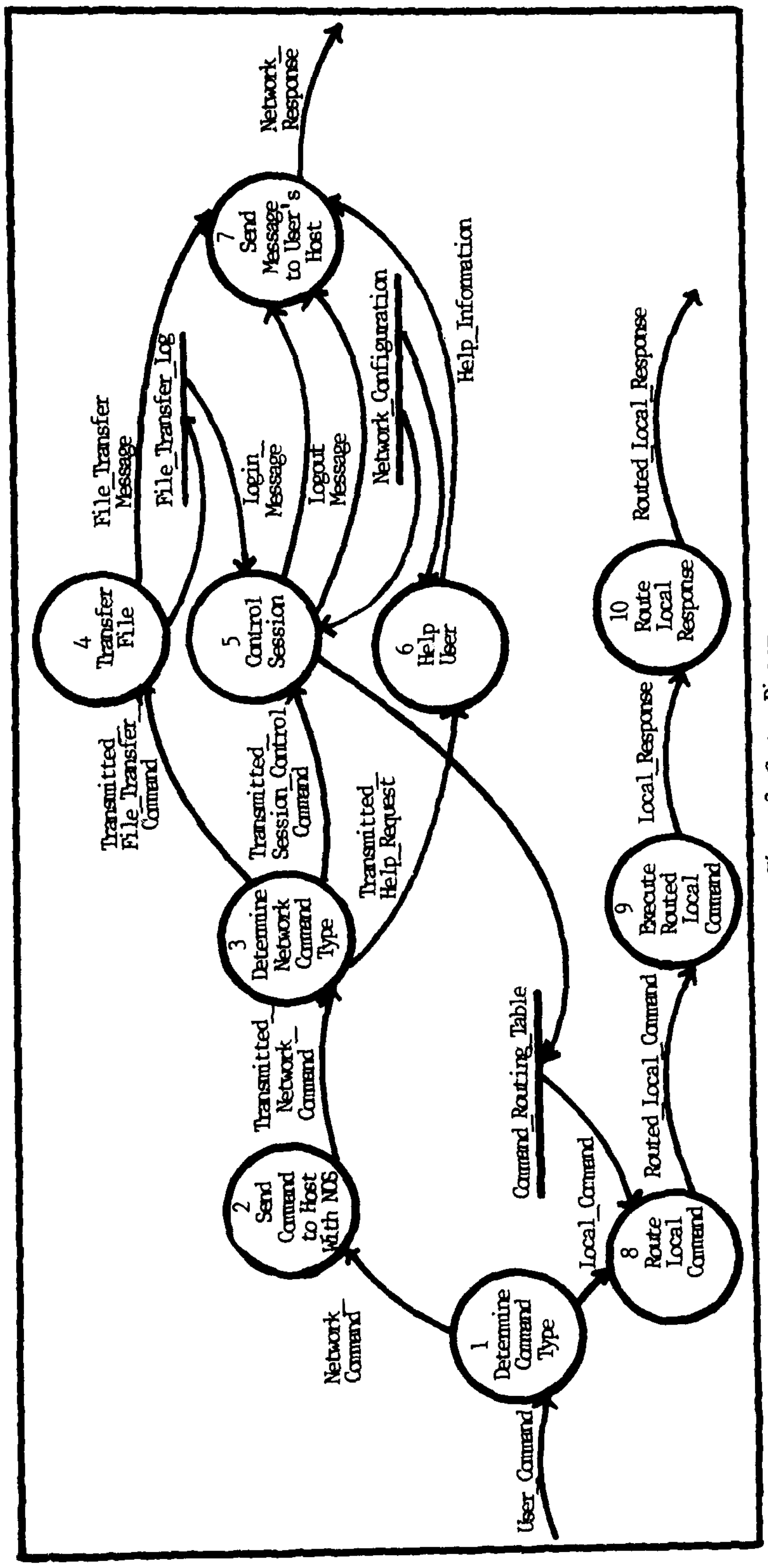


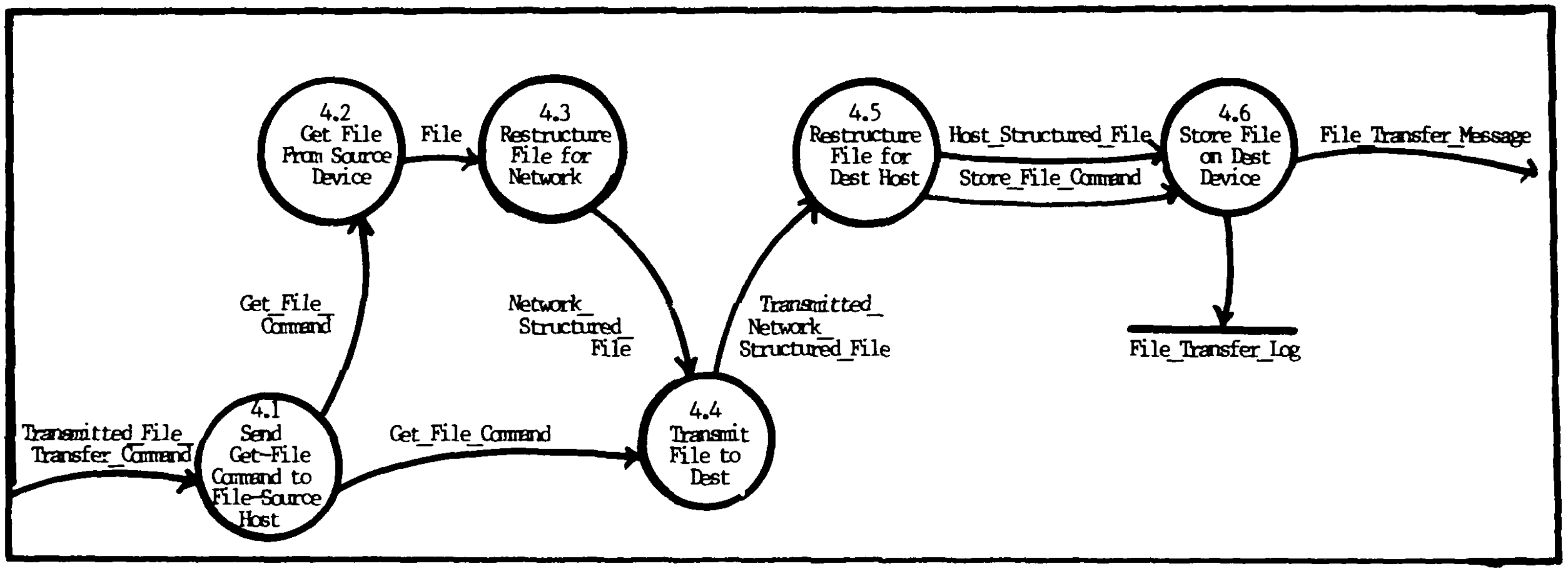
Figure 3 System Diagram

NOS must attempt to transfer the file and a file transfer message must be output and also entered into the file transfer log (4). Second, if the transmitted network command is to log into or log out of the network, then a log in or log out message must be output and the command routing table must be updated. The file transfer log and the network configuration file must both be accessed by the process (5). Third, if the transmitted network command is a help request, then the appropriate help information must be output to the user. Depending upon the type of help request, this process may require access to the network configuration file (6). Finally, the response that has been output from one of the three processes must be transmitted back to the user as the network response (7).

If the user command was a local command, then the command must be routed to the computer that the user specified in his log in command to DELNET. Since this does not have to be the host to which the user terminal is physically attached, the command routing table must be accessed to determine this routing (8). At the host to which the user is local, the command must be executed and a local response must be generated (9). Finally, this local response must be routed to the user (10). In processes 3, 4, 5, 6, and 9 there is also the possibility for commands to be invalid, and thus, error messages must be generated in place of the expected response.

File Transfer Requirements. The transfer file process was specified in more detail by the lower level DFD shown in Figure 4. At this level, the transmitted file transfer command must be parsed and the file transfer command parameters must be used to generate a get-file command that must be sent to the host that has the file to be transferred (4.1). The source host must then retrieve the file from the device that it is stored on and must restructure the file to a standard network file structure (4.2,4.3). Then the file must be transmitted to its destination using information also from the get-file command (4.4). Checkpointing must be used to transmit the file to insure that the entire file does not have to be retransmitted if an error should occur, but rather only the last portion to be transmitted. This is necessary to insure adequate throughput and response time. This checkpointing mechanism is shown in Figure 5. The file is broken into blocks which are individually transmitted (4.4.1,4.4.2). Thus, only those blocks not yet received correctly need to be transmitted when a file transmission error occurs. At the destination host, the file must be restructured for that host (4.5). Finally, the file must be stored on the destination device and the file transfer message output and entered into the file transfer log (4.6). There must be error messages generated if the file cannot be retrieved from the source device or if the destination device cannot store the file.

Session Control Requirements. The control session



4.1 Send Get-File Command to File-Source Host
4.2 Get File From Source Device
4.3 Restructure File for Network
4.4 Transmit File to Dest
4.5 Restructure File for Dest Host
4.6 Store File on Dest Device
Transmitted_File_Transfer_Command
Get_File_Command
Get_File_Command
File
Network_Structured_File
Transmitted_Network_Structured_File
Host_Structured_File
Store_File_Command
File_Transfer_Message
File_Transfer_Log

Figure 4 Transfer File (4.0) DFD

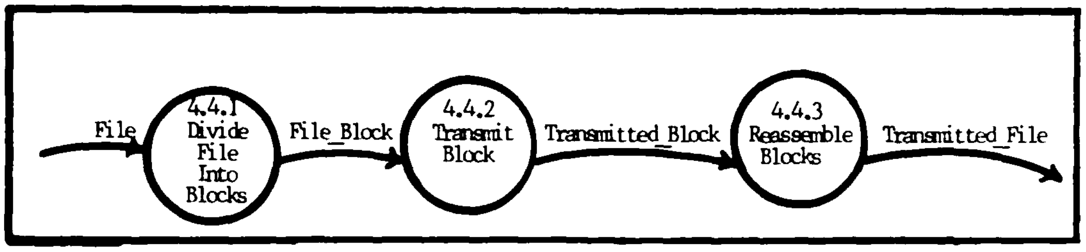


Figure 5 Transmit File (4.4) DFD

process may also be specified further and its lower level

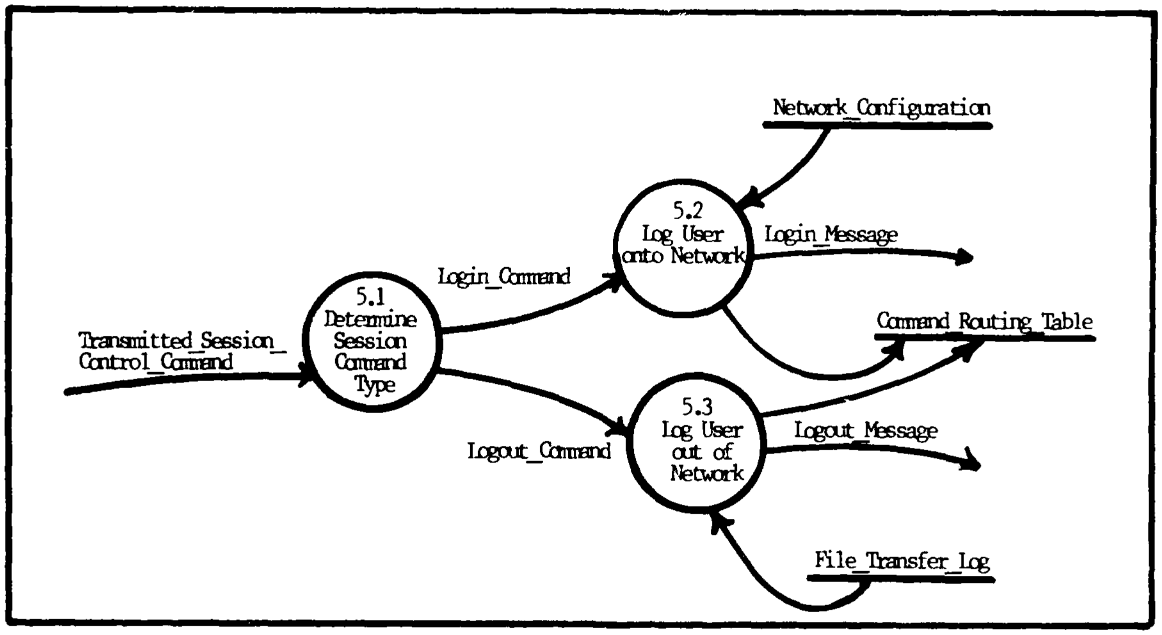
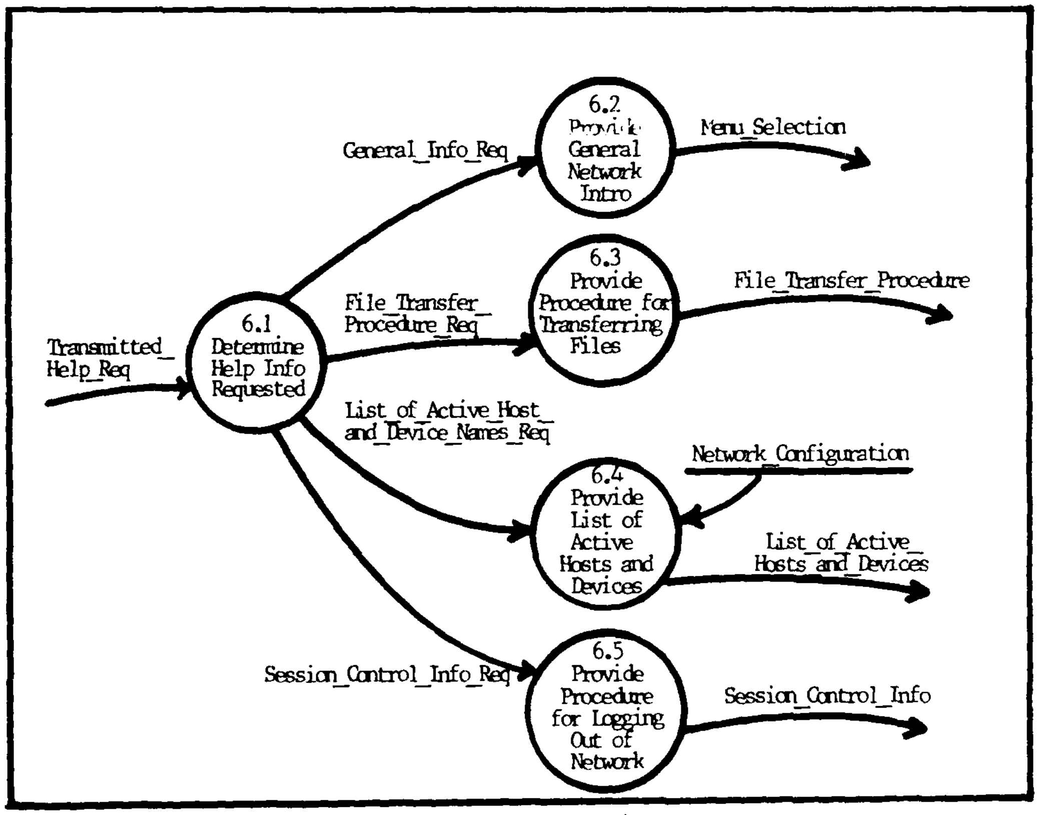


Figure 6 Control Session (5.0) DFD

DFD is shown in Figure 6. The transmitted session control command must be classified as either a log in command or a log out command (5.1). If it is a log in command, then the command must be validated by accessing the network configuration file to insure that the host exists in DELNET. Then a log in message must be output and the command routing table must be updated to show the local host for that user (5.2). If the transmitted session control command is a log out command, then the command routing table must be reset to the host to which the user is physically attached and a log out message output to the user display (5.3).

Help User Requirements. Finally, the help user process may be specified further by the lower level DFD



Transmitted_Help_Req
6.1 Determine Help Info Requested
General_Info_Req
6.2 Provide General Network Intro
Menu_Selection
File_Transfer_Procedure_Req
6.3 Provide Procedure for Transferring Files
File_Transfer_Procedure
List_of_Active_Host_and_Device_Names_Req
Network_Configuration
6.4 Provide List of Active Hosts and Devices
List_of_Active_Hosts_and_Devices
Session_Control_Info_Req
6.5 Provide Procedure for Logging Out of Network
Session_Control_Info

Figure 7 Help User (6.0) DFD

shown in Figure 7. The transmitted help request must be classified as either a general information request, a file transfer procedure request, a list of active host and device names request, or a session control information request (6.1). If it is a general information request, then a menu selection consisting of the available commands and their formats must be output along with the formats for the more specific help requests (6.2). If the transmitted help request is a file transfer procedure request, then the format for the transfer file command must be output (6.3). If the transmitted help request is a list of active hosts and device names request, then this list must be output by accessing the network configuration file (6.4). Finally, if the transmitted help request is a session control information request, then the information on the session control commands must be output (6.5).

Host-to-Host Protocol Requirements. The previous protocol requirements were specified to implement the applications of remotely logging into a computer and transferring files. These as well as the user help protocol rely upon a host-to-host transfer mechanism which was treated as a primitive by these processes. If the DFD partitioning process had been followed as specified by DeMarco, then the partitioning of the host-to-host transfer mechanism would have been duplicated several times. However, by defining the host-to-host transfer mechanism as a primitive that is used by several processes, it was possible to start a new set of DFDs for this mechanism without having to duplicate them for each process. Table 6 shows the overall process hierarchy for this set of DFDs.

The context diagram for the host-to-host protocol is shown in Figure 8. Obviously, the most important function of

Table 6

<u>Host-to-Host Protocol Process Hierarchy</u>

1. Execute Calling Host Protocol
    1.1 Divide Data Into Packets
    1.2 Place Channel in Data Transfer State
    1.3 Window Calling Host to Called Host Data Packet
    1.4 Send Packet to Calling Node
    1.5 Determine Calling Node to Calling Host Packet Type
    1.6 Determine Data Packet Category and Extract Flow Control
    1.7 Buffer Packet Sequence
    1.8 Assemble Packet Sequence
    1.9 Confirm Receipt of Interrupt Packet
    1.10 Send Data to Calling Host
2. Execute Calling Node Protocol
    2.1 Execute Calling Host Packet
        2.1.1 Determine Calling Host Packet Type
        2.1.2 Notify Called Node of Incoming Call
        2.1.3 Update Calling Node Windowing Variables
        2.1.4 Confirm Receipt of Calling Host Interrupt
        2.1.5 Confirm Calling Host Clear Packet
        2.1.6 Confirm Calling Host Reset Packet
        2.1.7 Confirm Calling Host Restart Packet
    2.2 Send Packet onto Network
    2.3 Execute Routed Called Node Packet
        2.3.1 Determine Called Node Packet Type
        2.3.2 Interrupt Calling Node to Calling Host Data Flow
        2.3.3 Notify Calling Host of Call Connection
        2.3.4 Window Called Host to Calling Host Data Packets
    2.4 Send Packet to Calling Host
    2.5 Recover from Procedure Error
3. Route Packets
4. Execute Called Node Protocol
    4.1 Execute Called Host Packet
        4.1.1 Determine Called Host Packet Type
        4.1.2 Relay Call Acceptance to Calling Node
        4.1.3 Update Called Node Windowing Variables
        4.1.4 Confirm Receipt of Called Host Interrupt
        4.1.5 Confirm Called Host Clear Request
        4.1.6 Confirm Called Host Reset Request
        4.1.7 Confirm Called Host Restart Request
    4.2 Send Packet onto Network
    4.3 Execute Routed Calling Node Packet
        4.3.1 Determine Calling Node Packet Type
        4.3.2 Interrupt Called Node to Called Host Data Flow
        4.3.3 Notify Called Host of Incoming Call
        4.3.4 Window Calling Host to Called Host Data Packets
    4.4 Send Packet to Called Host
    4.5 Recover from Procedure Error
5. Execute Called Host Protocol
    5.1 Divide Data into Packets
    5.2 Keep Channel in Data Transfer State
    5.3 Window Called Host to Calling Host Data Packet
    5.4 Send Packet to Called Node
    5.5 Determine Called Node to Called Host Packet Type
    5.6 Determine Data Packet Category and Extract Flow Control
    5.7 Buffer Packet Sequence
    5.8 Assemble Packet Sequence
    5.9 Confirm Receipt of Interrupt Packet
    5.10 Send Data to Called Host

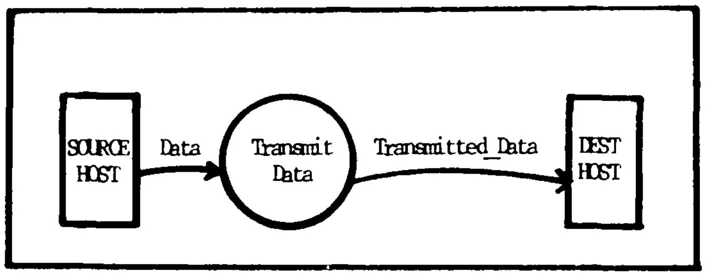


Figure 8 Host-to-Host Context Diagram

this protocol is to transmit data from the source host to the destination host. However, to accomplish this function reliably, there must be acknowledgement of the transferred data as well as error recovery procedures. Furthermore, that protocol must be transparent to the data since the data may be in ASCII, EBCDIC, or binary form. Since both long files and short network commands will be transmitted over the network; unless special precautions are taken, a long file transfer could easily degrade the response time of a network command to an unacceptable level. Thus, either file transfers must be able to be interrupted to allow network commands to be transmitted or separate mechanisms must be used for file transmission and network command transmission. The second option would require twice as many resources as the first alternative and so is not an acceptable alternative if the requirements can be met using the first option. The first option requires that the transmission medium be shared. Time division multiplexing (TDM) or frequency division multiplexing (FDM) could provide multiple paths between any two hosts as could many topologies. However, since the throughput requirement is much lower than the channel capacity between hosts on

DELNET, FDM was not a cost-effective way to meet the response time requirement. Also, restricting the topology to one having multiple paths between hosts conflicted with the flexibility requirement. Thus some form of TDM was needed. Packet-switching enables the communications path between two hosts to be shared by creating logical channels on the path. Each channel is limited to using the communications path only up to the time required to transmit a maximum-length packet. This prevents the network commands from being blocked by large file transfers and thus allows the response time requirement to be met.

Thus, a packet-switching protocol was required for DELNET. Due to time constraints on the development of DELNET, this protocol should be one that is a standard and thus, already specified. The alternatives for this protocol included the ARPANET protocol, IBM's SNA, and the X.25 protocol. Because the ARPANET protocol was the first packet-switching protocol, it does not employ the protocol advances such as the newer bit-oriented protocols for control of the links. IBM's SNA was not designed for heterogeneous computer networks and thus, is not appropriate for DELNET. For these reasons and also because of its versatility and endorsement by the International Consultative Committee on Telephones and Telegraphs (CCITT), the X.25 protocol was chosen for DELNET (Ref. 9: 243-284).

Overview of the X.25 Protocol. The X.25 protocol was developed to provide a transport mechanism for data access across a packet-switched network. It includes the three basic facilities of error control, flow control, and synchronization. There are three levels or layers within the X.25 protocol. Level 1 defines the physical protocol requirements and the link protocol is defined by level 2. The interface between hosts on the network with their entry nodes is specified by level 3 of the protocol. The software requirements for this protocol were then developed using a top-down approach by considering levels 3, 2, and 1 in that order.

X.25 Level 3 Requirements. The transmit data process shown in Figure 8 must be slightly modified for the X.25 protocol. This is because the X.25 protocol establishes virtual circuits and virtual calls between the hosts on the network and then allows a two-way exchange of data between the hosts. Thus, a more accurate context diagram of the

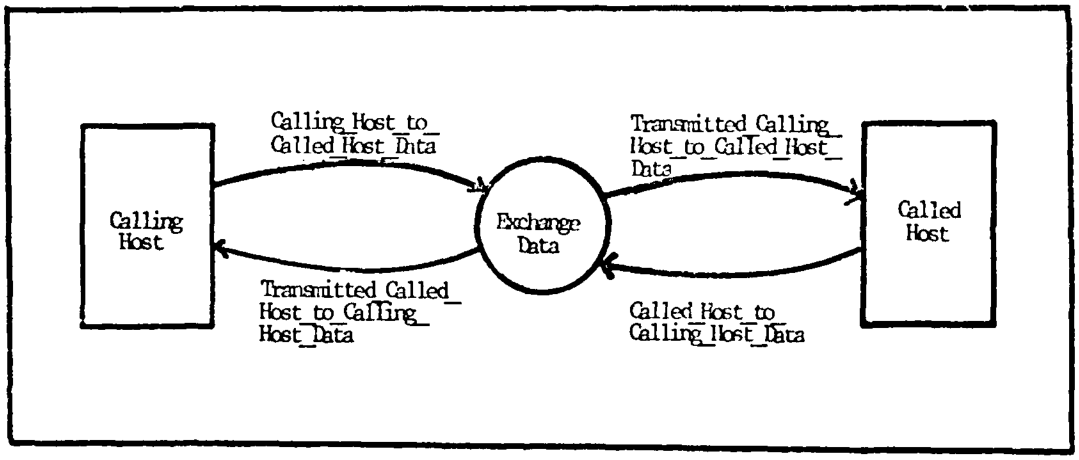


Figure 9 X.25 Level 3 Context Diagram

X.25 protocol is shown in Figure 9. The exchange data process is expressed in more detail by the X.25 Level 3 Overview DFD shown in Figure 10. To make the diagrams more readable, the terms "data terminal equipment" (DTE) and "data communications equipment" (DCE) used in the CCITT specification have been replaced by the terms "host" and "node" respectively. Processes 1, 2, 4, and 5 in Figure 10 are performed directly by the level 3 protocol while Process 3 uses a routing algorithm and the level 2 protocol to perform its function.

Execute Calling Host Protocol (1) Process. The lower-level DFD of the Execute Calling Host Protocol (1) process is shown in Figure 11. The data to be sent from the calling host must be divided into 128-byte packets (1.1). These packets must then be queued to be transmitted to the called host over the X.25 channel. If a virtual circuit exists between the calling and called hosts, then the X.25 channel that the circuit is on must be placed in the data transfer state by issuing a reset request packet and waiting for receipt of a reset confirmation packet from the calling node. If no channel has been allocated as a virtual circuit between the two hosts, then a virtual call must be made. The calling host must send a call request packet to the calling node. Then upon receipt of a call connected packet from the calling node, the virtual call has been made and the X.25 channel over which the call was made is in the data transfer state.

To clear a virtual call when all the data from the

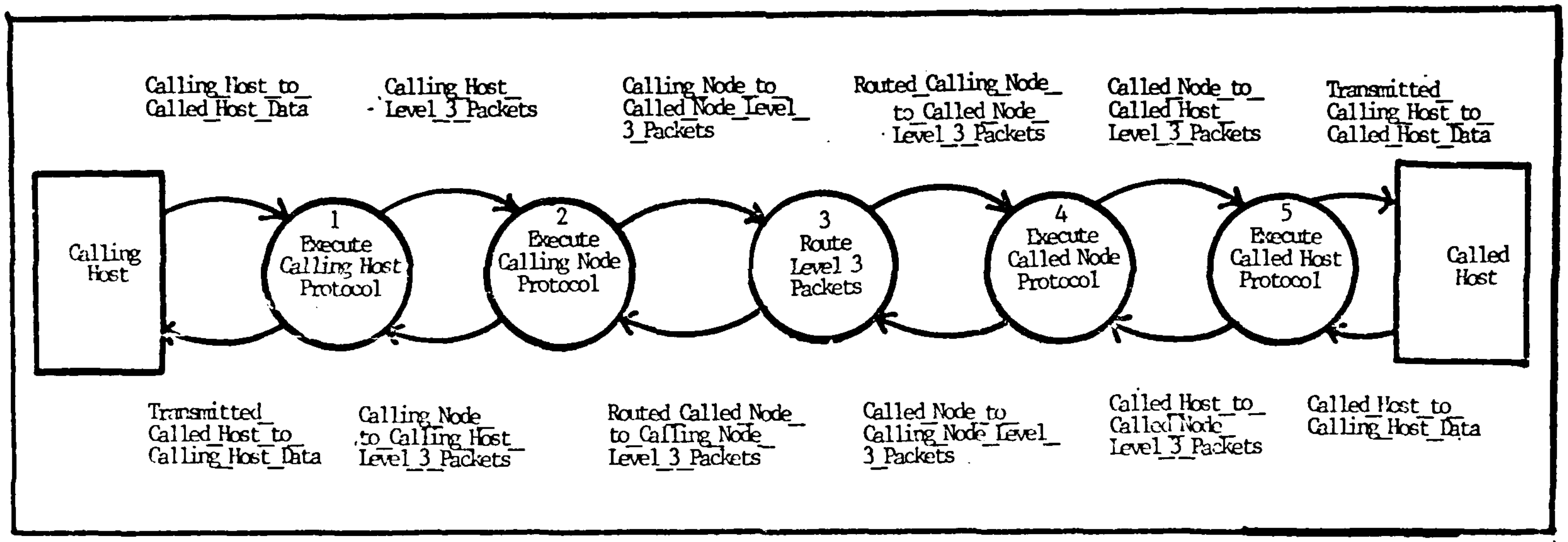
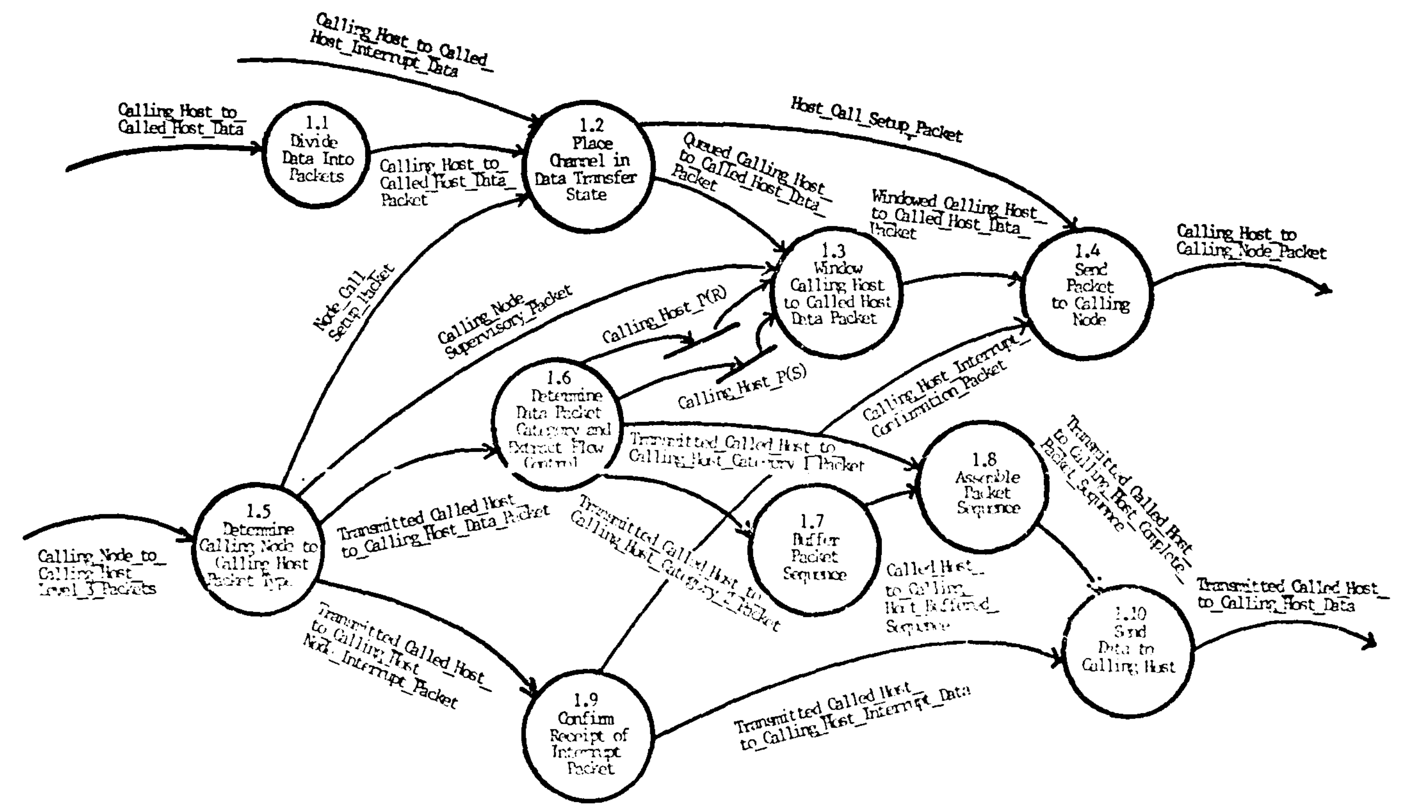

Calling Host
1 Execute Calling Host Protocol
2 Execute Calling Node Protocol
3 Route Level 3 Packets
4 Execute Called Node Protocol
5 Execute Called Host Protocol
Called Host
Calling_Host_to_Called_Host_Data
Calling_Host_Level_3_Packets
Calling_Node_to_Called_Node_Level_3_Packets
Routed_Calling_Node_to_Called_Node_Level_3_Packets
Called_Node_to_Called_Host_Level_3_Packets
Transmitted_Calling_Host_to_Called_Host_Data
Transmitted_Called_Host_to_Calling_Host_Data
Calling_Node_to_Calling_Host_Level_3_Packets
Routed_Called_Node_to_Calling_Node_Level_3_Packets
Called_Node_to_Calling_Node_Level_3_Packets
Called_Host_to_Called_Node_Level_3_Packets
Called_Host_to_Calling_Host_Data

Figure 10 X.25 Level 3 Overview DFD



1.1 Divide Data Into Packets
1.2 Place Channel in Data Transfer State
1.3 Window Calling Host to Called Host Data Packet
1.4 Send Packet to Calling Node
1.5 Determine Calling Node to Calling Host Packet Type
1.6 Determine Data Packet Category and Extract Flow Control
1.7 Buffer Packet Sequence
1.8 Assemble Packet Sequence
1.9 Confirm Receipt of Interrupt Packet
1.10 Send Data to Calling Host
Calling_Host_to_Called_Host_Data
Calling_Host_to_Called_Host_Interrupt_Data
Calling_Host_to_Called_Host_Data_Packet
Host_Call_Setup_Packet
Queued_Calling_Host_to_Called_Host_Data_Packet
Windowed_Calling_Host_to_Called_Host_Data_Packet
Calling_Host_to_Calling_Node_Packet
Node_Call_Setup_Packet
Calling_Node_Supervisory_Packet
Calling_Host_P(R)
Calling_Host_P(S)
Calling_Host_Interrupt_Confirmation_Packet
Transmitted_Called_Host_to_Calling_Host_Category_1_Packet
Transmitted_Called_Host_to_Calling_Host_Complete_Packet_Sequence
Calling_Node_to_Calling_Host_Level_3_Packets
Transmitted_Called_Host_to_Calling_Host_Data_Packet
Transmitted_Called_Host_to_Calling_Host_Category_2_Packet
Called_Host_to_Calling_Host_Buffered_Sequence
Transmitted_Called_Host_to_Calling_Host_Data
Transmitted_Called_Host_to_Calling_Host_Node_Interrupt_Packet
Transmitted_Called_Host_to_Calling_Host_Interrupt_Data

Figure 11 Execute Calling Host Protocol (1) DFD

called host to the calling host has been transmitted, the calling host must send a clear request to the calling node. The call must be cleared and the channel made available when the calling host receives a clear confirmation packet from the calling node.

If a local procedure error occurs, the virtual calls and virtual circuits must be reset as necessary to recover from the error. The calling host must send a reset request packet to the calling node for each logical channel to be reset. The channels will be reset upon receipt of a reset confirmation packet from the calling node. If the entire calling-host-to-calling-node interface is to be reinitialized with all virtual calls being cleared and all virtual circuits being reinitialized, then the calling host must send a restart request packet to the calling node. The interface must then be reinitialized upon receipt of a restart confirmation packet.

The state of the channel must also be able to affected by the calling node. If the calling host is waiting to receive a call connected packet from the calling node and instead receives a clear indication packet from the calling node, then the calling host must send a clear confirmation packet back to the calling node and must clear the pending call request. Also, if the calling host receives a reset indication packet from the calling node, then the calling host must reset the appropriate channel and send a reset confirmation packet back to the calling node. Finally, the calling host must clear all virtual calls and reset all virtual circuits upon receipt of a restart indication packet and must respond with a restart confirmation packet back to the calling node (1.2).

There must be the capability from the calling host to control the flow of data packets to the calling node as well as the capability to interrupt this flow control with data interrupt packets. This must be accomplished by the Window Calling Host to Called Host Data Packet (1.3) process and the Send Packet to Calling Node (1.4) process. A windowing mechanism must be implemented using the Packet Send Variable (P(S)) and the Packet Receive Variable (P(R)) which must allow up to W data packets that were transmitted to the calling node to be pending acknowledgement at any time. W is the window size. The packets must be numbered sequentially modulo 8 and the lower window edge (last packet acknowledged by the calling node) must be updated. This must be done using supervisory packets from the calling node as well as the flow control acknowledgement field in the transmitted called host to calling host data packets (1.3). The windowed calling host data packets must also carry an acknowledgement in its flow control acknowledgement field of the last called host to calling host data packet received by the calling host. Since the host interrupt packets to the calling node are not windowed, they must be acknowledged by an interrupt confirmation packet from the calling node (1.4).

The calling host protocol must also accept data packets from the called host that are relayed by the calling node. These packets may be either data interrupt packets from the called host or they may be data packets that were windowed by the calling node protocol (1.5). The data interrupt packets must be confirmed as already mentioned by transmitting an interrupt confirmation packet back to the calling node. Also, the interrupt data must be extracted from the packet and sent to the calling host (1.9,1.10). The windowed data packets from the calling node must be reassembled into the packet sequences that comprised the original data from the called host. These packet sequences must be assembled by first buffering all packets which indicate that more packets follow in the sequence (category 2 packets) until a packet arrives that indicates that it is the last packet in the sequence (category 1 packet). This category 1 packet must then trigger the release of the complete packet sequence (1.6,1.7,1.8). Finally, this called host to calling host complete packet sequence must be sent to the calling host (1.10).

Execute Calling Node Protocol Process. Process 2 can also be expanded into a lower-level DFD and its DFD is shown in Figure 12. The level 3 packets from the calling host must be executed and must result in packets being sent out on the network to be routed to the called node as well as resulting in packets being sent back to the calling host (2.1,2.2). Flow control information from the calling host

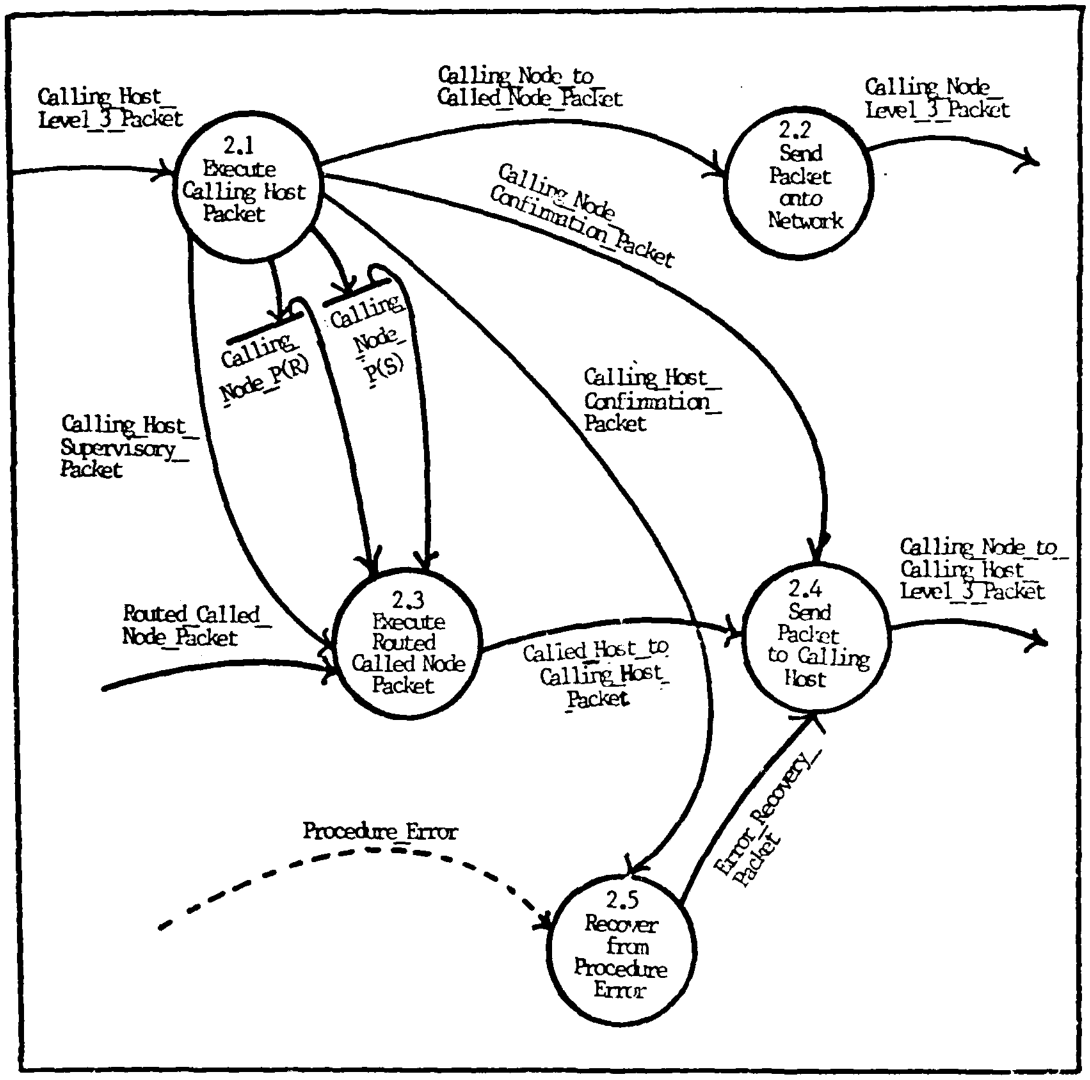
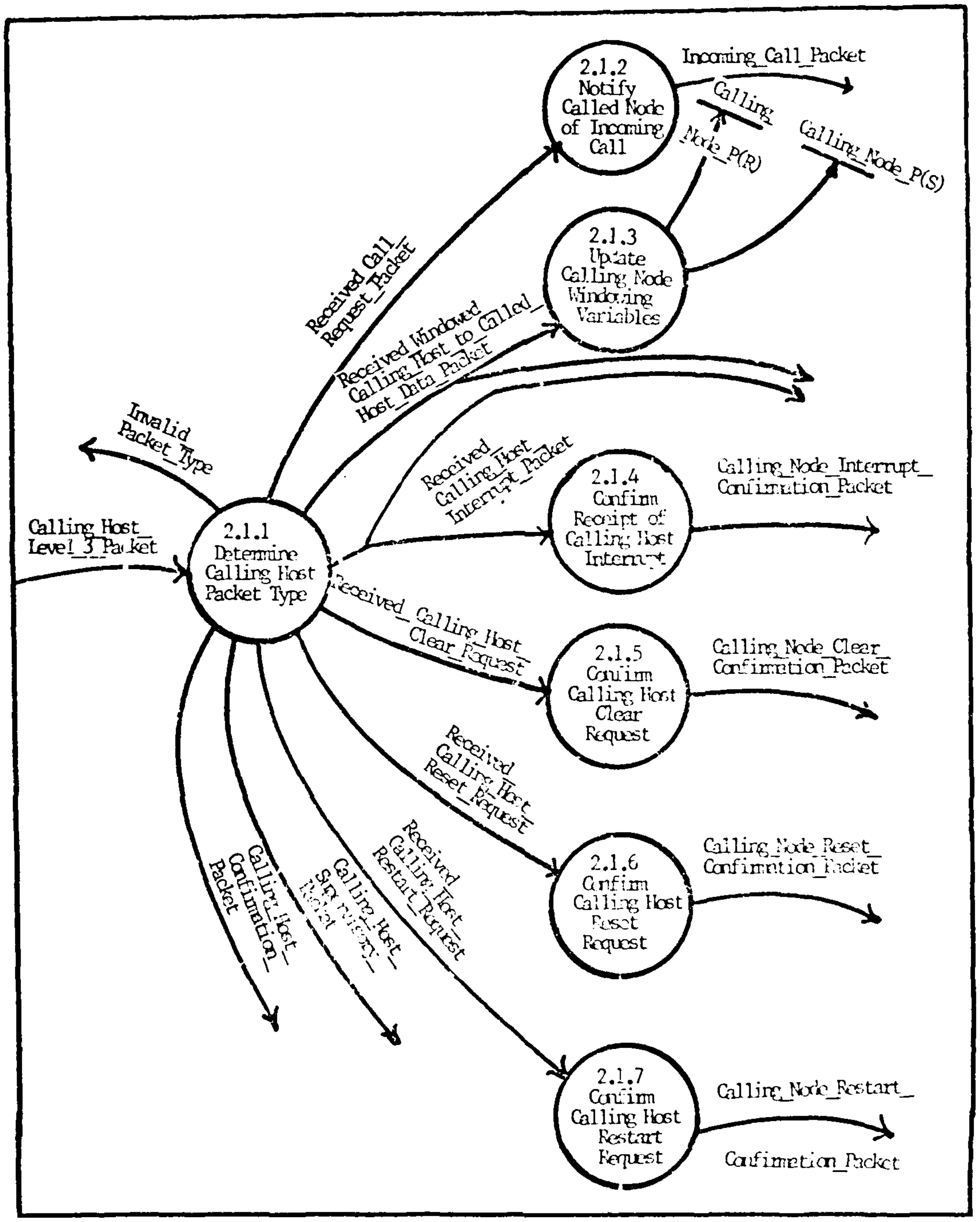


Figure 12 Execute Calling Node Protocol (2) DFD

level 3 packets must be used to update the calling node windowing variables (2.1).

The routed called node packets must also be executed and they must result in packets being sent to the calling host (2.3,2.4). The windowing variables as well as the calling host supervisory packets must be used for flow control (2.3). Finally, there must be a process to recover from a local procedure error that might occur in one of the other processes. This recovery must be initiated by either a reset indication packet or a restart indication packet sent from the calling node to the calling host (2.5).

An expansion of the execute calling host packet into a lower-level DFD is shown in Figure 13. The calling host level 3 packet must first be classified as one of the types of packets shown in the DFD (2.1.1). If a call request packet is received, then an incoming call packet must be generated (2.1.2). If the received packet is a calling host to called host data packet, then its flow control acknowledgements must be used to update the calling node windowing variables (2.1.3). Both the incoming call packet and the calling host to called host data packet as well as a calling host interrupt packet that is received must be sent out on the network. The interrupt packet must, in addition, be confirmed by an interrupt confirmation packet from the calling node (2.1.4). If a clear request is received, then it must be confirmed by a clear confimation packet from the calling node (2.1.5). A node reset confirmation packet must be generated if a reset request is received and the channel must be reset at the calling node (2.1.6). Likewise, receipt of a restart request packet must result in the reinitialization of the calling-node-to-calling-host interface and the reinitialization must be confirmed with a restart confirmation packet (2.1.7). All of these confirmation packets must be sent back to the calling host



2.1.1 Determine Calling Host Packet Type
2.1.2 Notify Called Node of Incoming Call
2.1.3 Update Calling Node Windowing Variables
2.1.4 Confirm Receipt of Calling Host Interrupt
2.1.5 Confirm Calling Host Clear Request
2.1.6 Confirm Calling Host Reset Request
2.1.7 Confirm Calling Host Restart Request
Calling_Host_Level_3_Packet
Invalid Packet_Type
Received_Call_Request_Packet
Received_Windowed_Calling_Host_to_Called_Host_Data_Packet
Received_Calling_Host_Interrupt_Packet
Received_Calling_Host_Clear_Request
Received_Calling_Host_Reset_Request
Received_Calling_Host_Restart_Request
Calling_Host_Confirmation_Packet
Calling_Host_Supervisory_Packet
Incoming_Call_Packet
Calling_Node_P(R)
Calling_Node_P(S)
Calling_Node_Interrupt_Confirmation_Packet
Calling_Node_Clear_Confirmation_Packet
Calling_Node_Reset_Confirmation_Packet
Calling_Node_Restart_Confimation_Packet

Figure 13 Execute Calling Host Packet (2.1) DFD

and together, they constitute the calling node confirmation packet data flow shown in Figure 12. A confirmation packet from the calling host on an action initiated by the calling node must be an input to the Recover From Procedure Error (1.5) process. This is shown in Figure 12 by the calling host confirmation packet data flow. Finally, a calling host supervisory packet must be an input to the Execute Routed Called Node Packet (1.3) process in Figure 12.

The process that executes routed packets from the called node is specified in more detail through the lower-level DFD in Figure 14. A routed packet from the

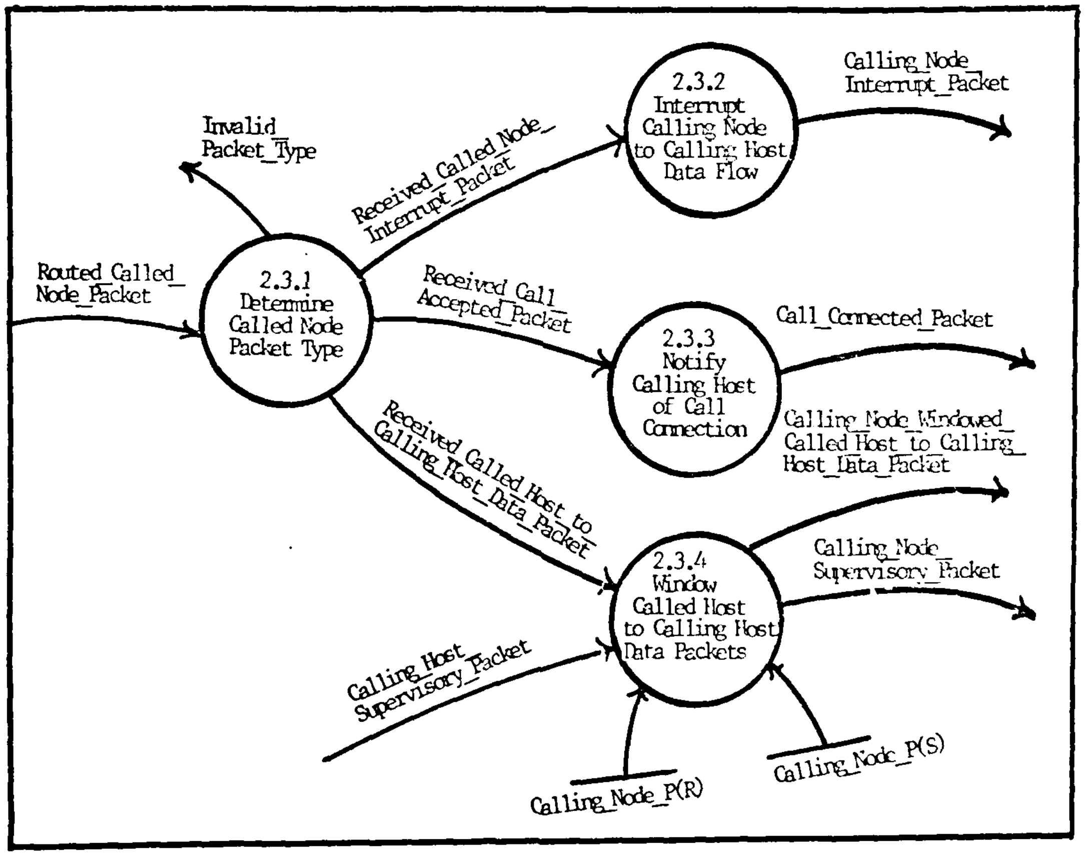

Routed_Called_ Node_Packet
2.3.1 Determine Called Node Packet Type
Invalid_ Packet_Type
Received_Called_Node_ Interrupt_Packet
2.3.2 Interrupt Calling Node to Calling Host Data Flow
Calling_Node_ Interrupt_Packet
Received_Call_ Accepted_Packet
2.3.3 Notify Calling Host of Call Connection
Call_Connected_Packet
Received_Called_Host_to_ Calling_Host_Data_Packet
2.3.4 Window Called Host to Calling Host Data Packets
Calling_Node_Windowed_ Called_Host_to_Calling_ Host_Data_Packet
Calling_Node_ Supervisory_Packet
Calling_Host_ Supervisory_Packet
Calling_Node_P(R)
Calling_Node_P(S)

Figure 14 Execute Routed Called Node Packet (2.3) DFD

called node must first be classified as either a node interrupt packet, a call accepted packet, or a called host to calling host data packet (2.3.1). If it is not one of these, then it must be considered an invalid packet and discarded. A called node interrupt packet that is received must interrupt the flow of data from the calling node to the calling host. Then, the data contained in the interrupt packet must be sent to the calling host in another interrupt packet from the calling node (2.3.2). If the routed called node packet is a call accepted packet, then the calling host must be notified of the virtual call connection by sending it a call connected packet (2.3.3). A called host to calling host data packet must be windowed and sent to the calling host. The windowing is accomplished by using the Receive Ready (RR) and Receive Not Ready (RNR) supervisory packets from the calling host and the calling node windowing variables. These are used to send both windowed called host to calling host data packets as well as calling node supervisory packets to the calling host (2.3.4).

Execute Called Node Protocol Process. The Execute Called Node Protocol (4) process shown in Figure 10 is very similar to the execute calling node protocol process. The lower-level DFD's for this process are shown in Figures 15, 16, and 17. As can be seen from these DFD's, the requirements for the Execute Called Node Protocol (4) process are almost identical to those for the Execute Calling Node Protocol (2) process. The major difference is in the handling of the virtual call setup. The called node must relay the incoming call packet to the called host and

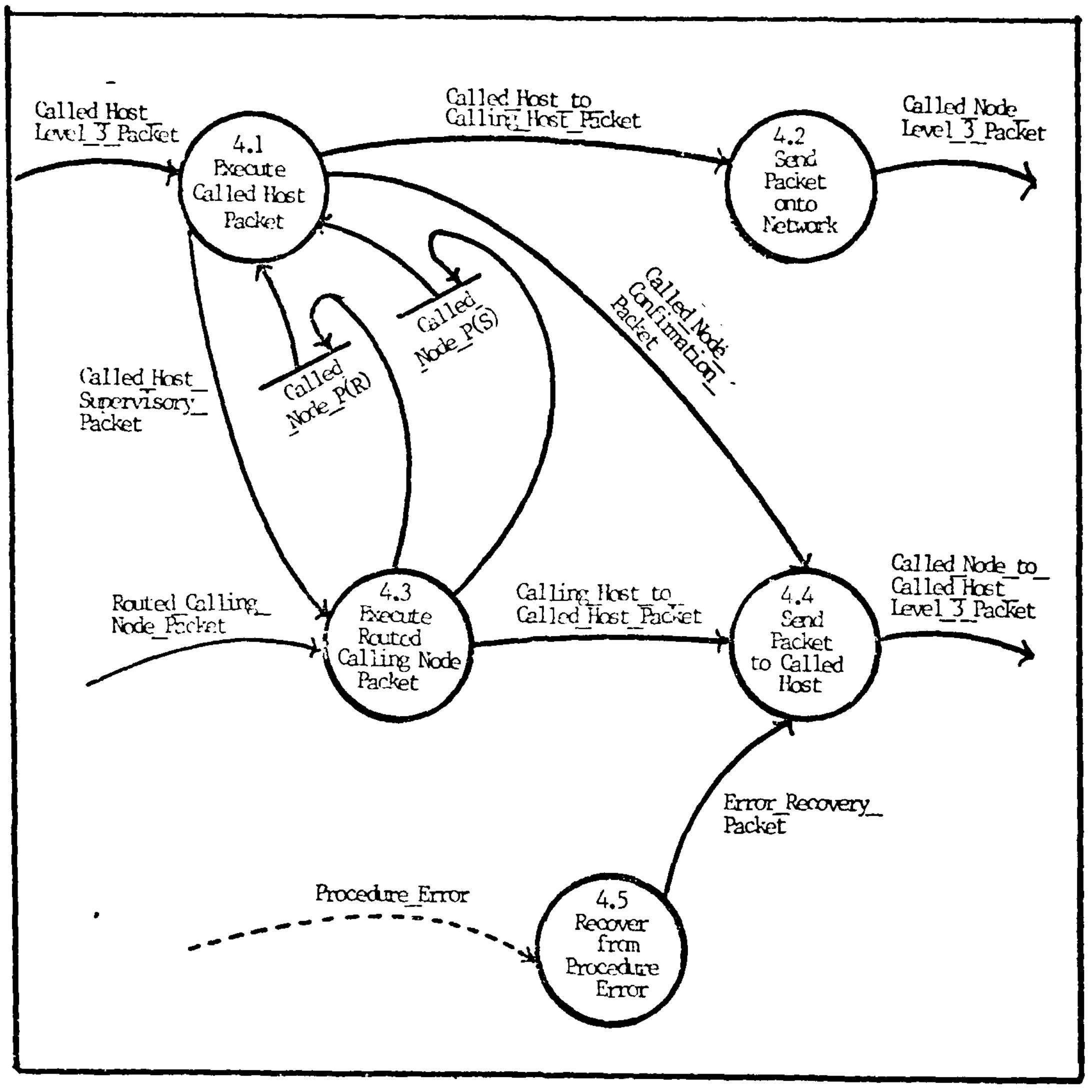

Called Host Level_3_Packet
4.1 Execute Called Host Packet
Called Host to Calling Host Packet
4.2 Send Packet onto Network
Called Node Level_3_Packet
Called Node Confirmation Packet
Called Node_P(S)
Called Node_P(R)
Called Host Supervisory Packet
Routed Calling Node Packet
4.3 Execute Routed Calling Node Packet
Calling Host to Called Host Packet
4.4 Send Packet to Called Host
Called Node to Called Host Level_3_Packet
Error_Recovery_Packet
Procedure_Error
4.5 Recover from Procedure Error

Figure 15 Execute Called Node Protocol (4) DFD

then must relay the call accepted packet from the called host to the calling node (4.1.2,4.3.3).

Execute Called Host Protocol Process. As with the Execute Called Node Protocol (4) process, the Execute Called Host Protocol (5) process also has an almost identical counterpart. As can be seen from the lower-level DFD of the Execute Called Host Protocol (5) shown in Figure 18, the

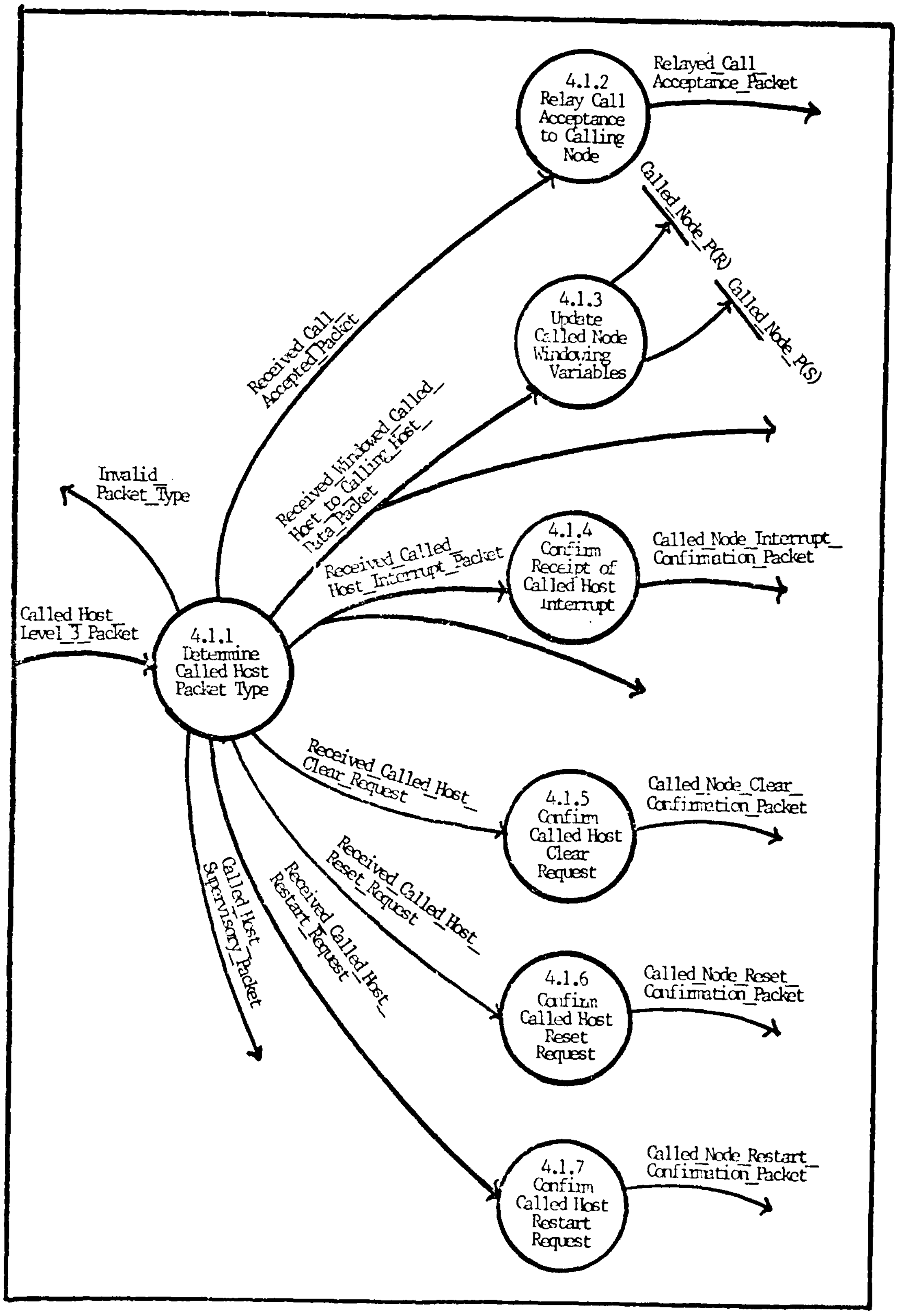

4.1.1 Determine Called Host Packet Type
4.1.2 Relay Call Acceptance to Calling Node
4.1.3 Update Called Node Windowing Variables
4.1.4 Confirm Receipt of Called Host Interrupt
4.1.5 Confirm Called Host Clear Request
4.1.6 Confirm Called Host Reset Request
4.1.7 Confirm Called Host Restart Request
Called_Host_Level_3_Packet
Invalid_Packet_Type
Received_Call_Accepted_Packet
Relayed_Call_Acceptance_Packet
Called_Node_P(R)
Called_Node_P(S)
Received_Windowed_Called_Host_to_Calling_Host_Data_Packet
Received_Called_Host_Interrupt_Packet
Called_Node_Interrupt_Confirmation_Packet
Received_Called_Host_Clear_Request
Called_Node_Clear_Confirmation_Packet
Received_Called_Host_Reset_Request
Called_Node_Reset_Confirmation_Packet
Received_Called_Host_Restart_Request
Called_Node_Restart_Confirmation_Packet
Called_Host_Supervisory_Packet

Figure 16 Execute Called Host Packet (4.1) DFD

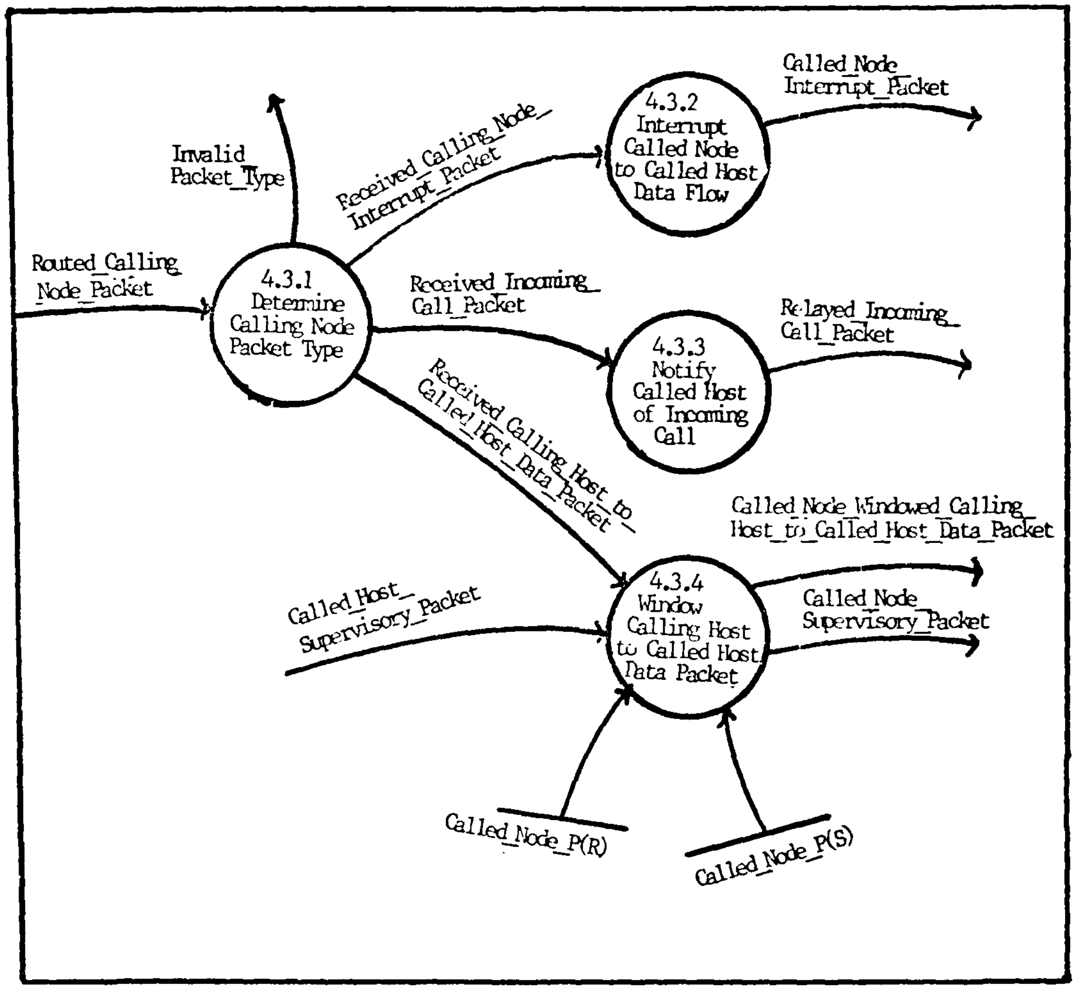
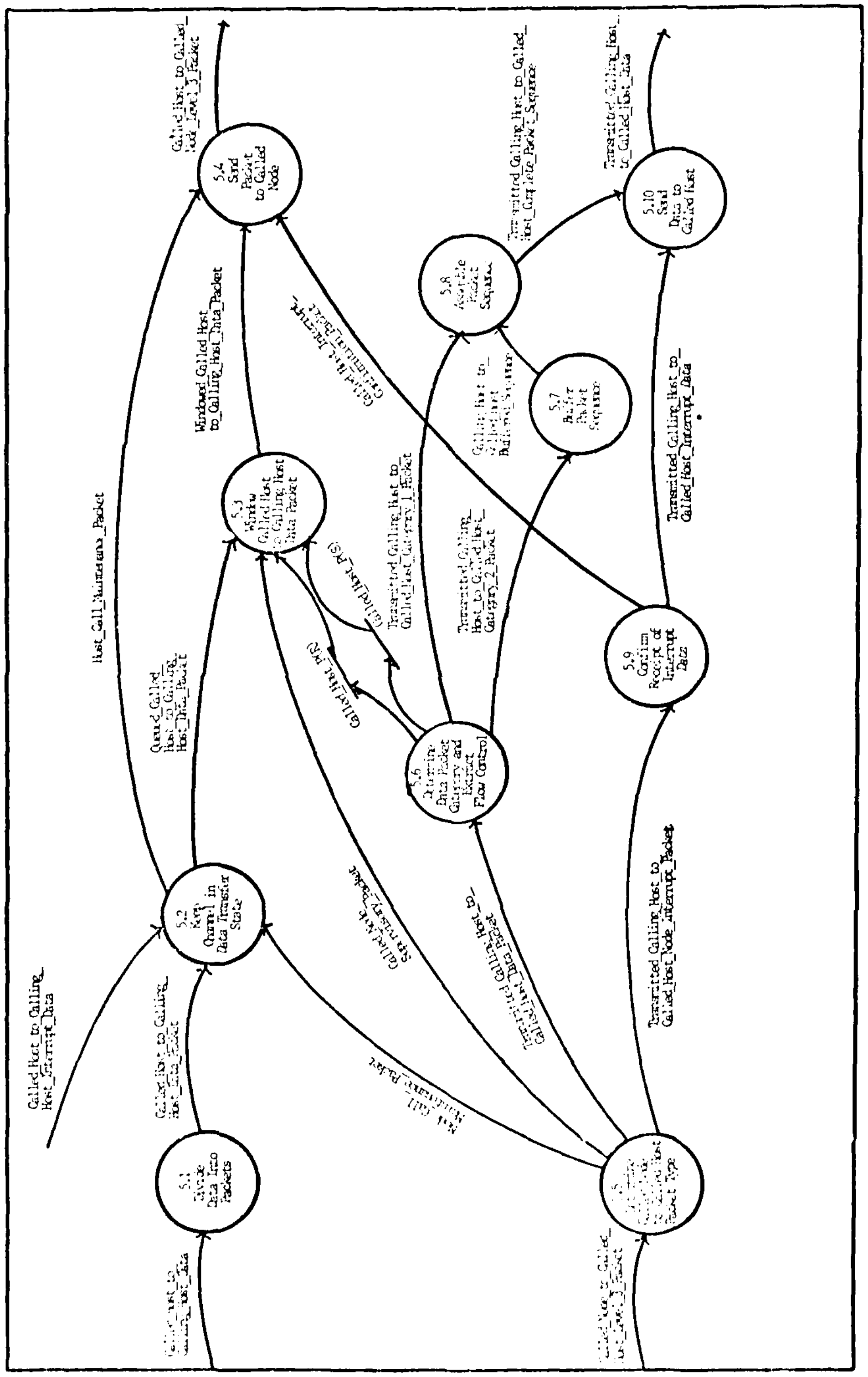


Figure 17 Execute Routed Calling Node Packet (4.3) DFD

principal difference is again in handling the virtual call. The Execute Called Host Protocol (5) must accept a virtual call by sending a call accepted packet to the called node and it cannot by definition generate a call request (5.2).

Network Protocol Requirements. As with the host-to-host transfer mechanism, the network transfer mechanism is used by several processes and is treated as a primitive in the host-to-host protocol. It is also



5.1 Divide Data Into Packets
5.2 Keep Channel in Data Transfer State
5.3 Window Called Host to Calling Host Data Packet
5.4 Send Packet to Called Node
5.6 Determine Data Packet Category and Extract Flow Control
5.7 Buffer Packet Sequence
5.8 Assemble Packet Sequence
5.9 Confirm Receipt of Interrupt Data
5.10 Send Data to Called Host
Called_Host_to_Calling_Host_Data
Called_Host_to_Calling_Host_Interrupt_Data
Called_Host_to_Calling_Host_Data_Packet
Host_Call_Maintenance_Packet
Queued_Called_Host_to_Calling_Host_Data_Packet
Windowed_Called_Host_to_Calling_Host_Data_Packet
Called_Host_to_Called_Node_Level_3_Packet
Called_Host_Interrupt_Confirmation_Packet
Called_Host_P(S)
Called_Host_P(R)
Transmitted_Calling_Host_to_Called_Host_Category_1_Packet
Transmitted_Calling_Host_to_Called_Host_Category_2_Packet
Calling_Host_to_Called_Host_Buffered_Sequence
Transmitted_Calling_Host_to_Called_Host_Complete_Packet_Sequence
Transmitted_Calling_Host_to_Called_Host_Data
Transmitted_Calling_Host_to_Called_Host_Interrupt_Data
Transmitted_Calling_Host_to_Called_Host_Node_Interrupt_Packet
Transmitted_Calling_Host_to_Called_Host_Data_Packet
Called_Host_Supervisory_Packet
Host_Call_Maintenance_Packet

Figure 18 Execute Called Host Protocol (5) DFD

specified by a set of DFD's starting with the context

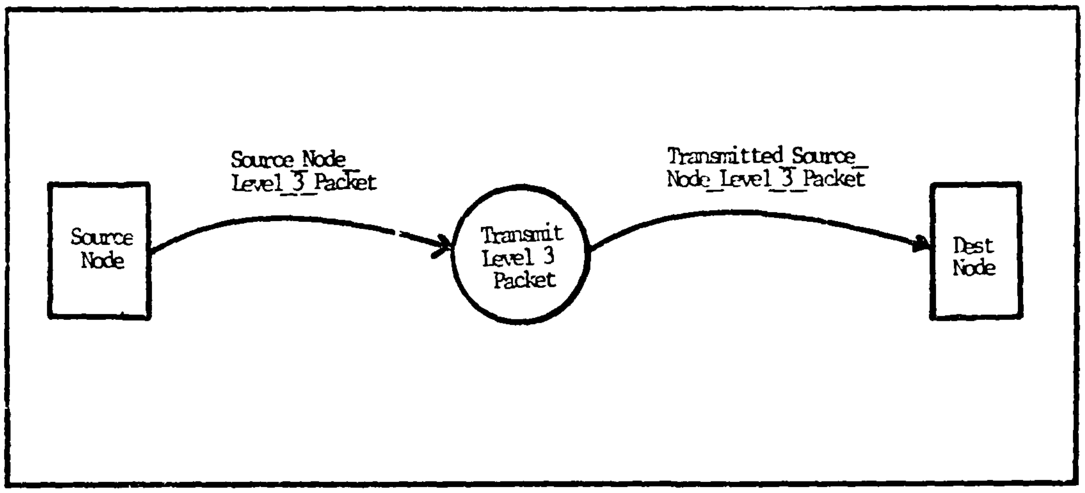


Figure 19. Network Protocol Context Diagram

diagram shown in Figure 19.

The network protocol, then, must transmit packets from their source node to their destination node. As can be seen from the Network Protocol Overview DFD in Figure 20, there

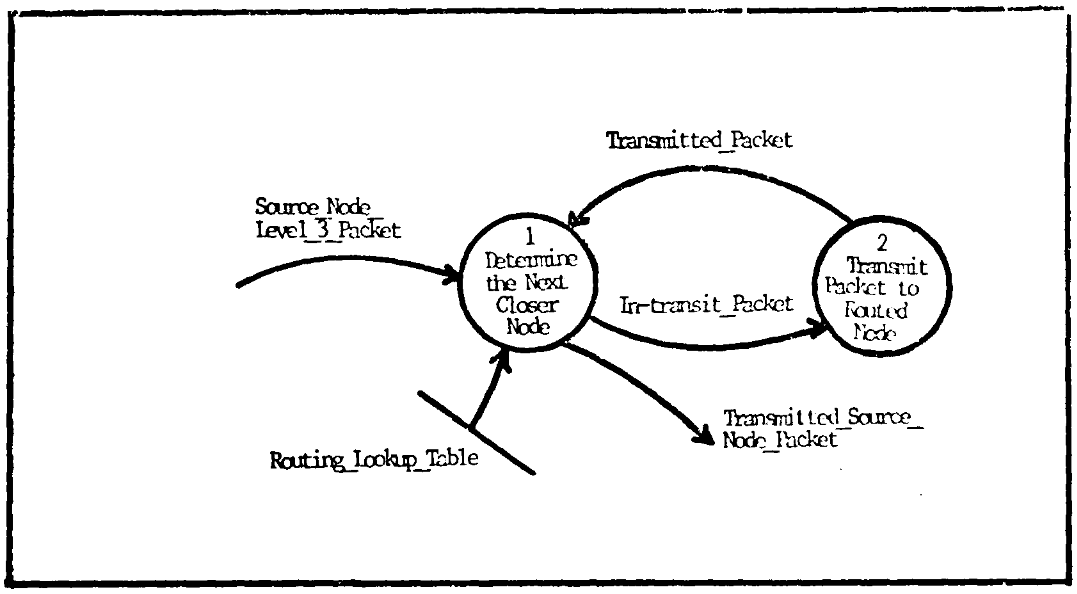


Figure 20 Network Protocol Overview DFD

are actually two processes required to accomplish the network protocol. First, the packet must be routed to a closer node using a lookup table unless it has reached its destination node (1). If it is not at the destination node, then the in-transit packet must be sent to the closer node to which it has been routed (2). Thus, both a routing algorithm to implement the first process and a link transfer mechanism to implement the second process are required for the network protocol.

Routing Algorithm Requirements. Given a simple network topology, the routing algorithm may also be very simple. There is only a requirement for a look up table in each node that is initialized when the network is booted. Since the availability requirement was only 90 percent, there is no need to make the routing algorithm adaptive; however error messages must be given to the user if a host active on the network cannot be reached. If a more complex topology is used, then a distributed adaptive routing algorithm should be implemented as recommended by Ravenscroft (Ref. 16).

Link Protocol Requirements. The X.25 protocol requires that the High Level Data Link Control (HDLC) be used for the level 2 or link protocol. The overview DFD for this protocol is shown in Figure 21. Also, Table 7 shows the process hierarchy for this set of DFDs. This protocol must specify the mechanism by which information is exchanged between adjacent nodes in the network. The Execute HDLC

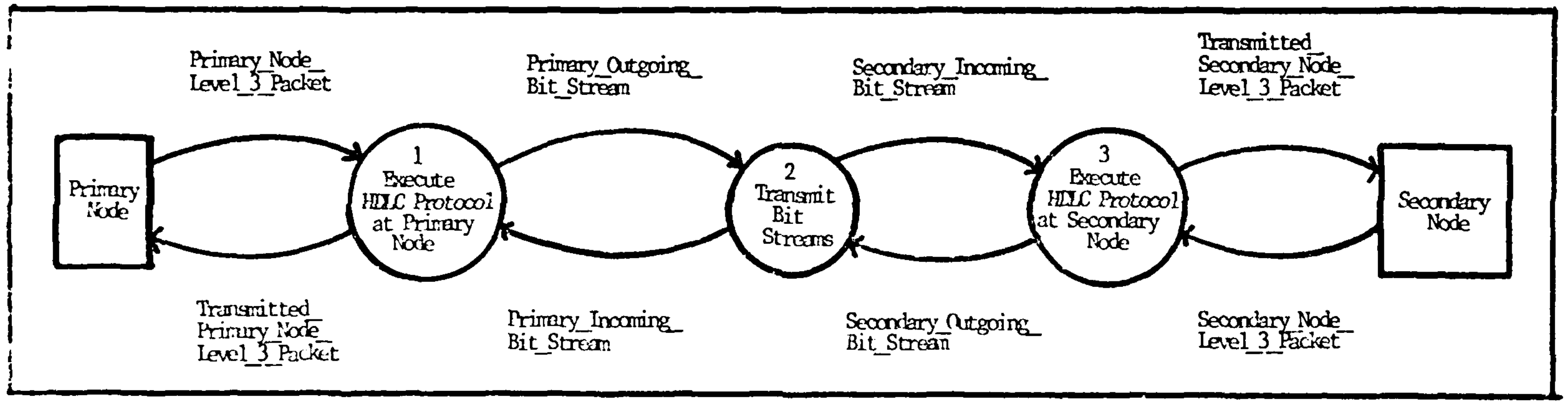


Figure 21 X.25 Level 2 (HDLC) Overview DFD

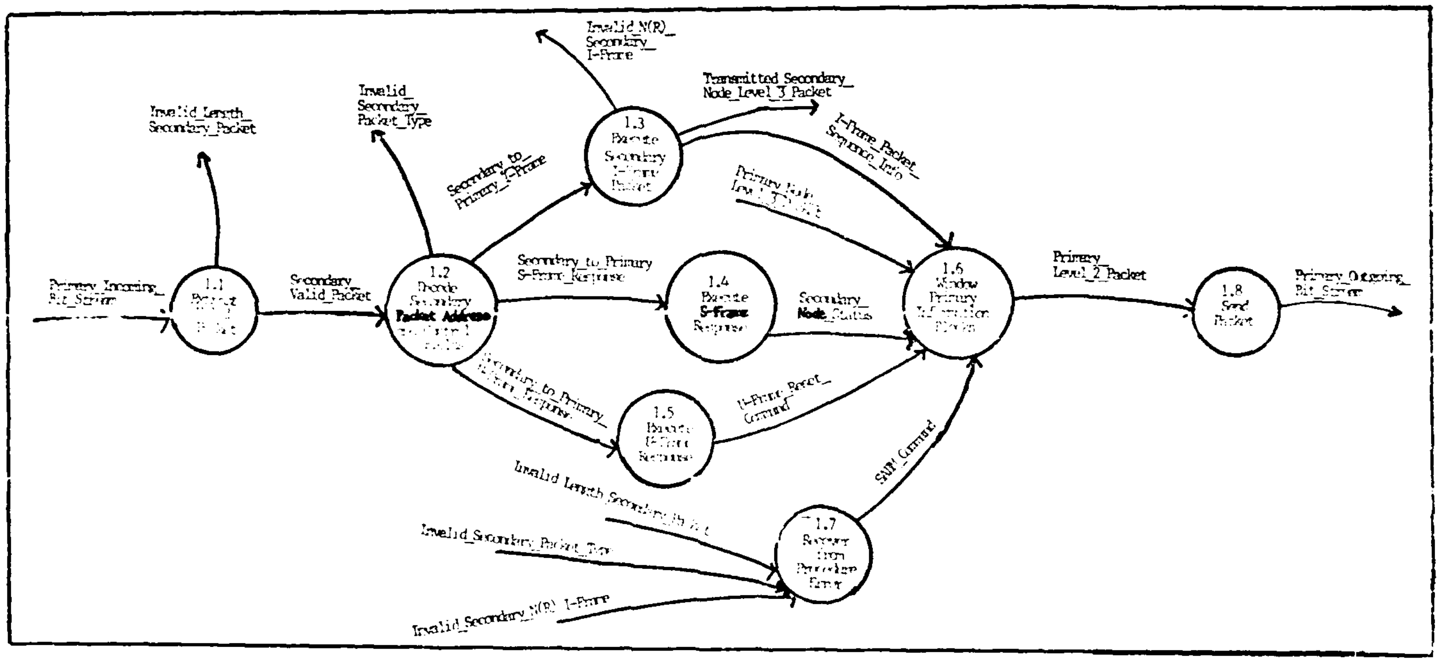
Table 7

Link Protocol Process Hierarchy

```
1.  Execute HDLC Protocol at Primary Node
    1.1  Extract Valid Packet
         1.1.1  Extract Information Between Flags
         1.1.2  Unstuff Zeros
         1.1.3  Check Length
         1.1.4  Generate CRC Remainder
         1.1.5  Check FCS Block
    1.2  Decode Secondary Address and Control Fields
    1.3  Execute Secondary I-Frame Packet
         1.3.1  Validate I-Frame Packet
         1.3.2  Parse I-Frame
         1.3.3  Parse I-Frame Control Field
    1.4  Execute S-Frame Response
         1.4.1  Parse S-Frame Response Control Field
         1.4.2  Decode Response Supervisory Bits
         1.4.3  Send Ready Status and Extract N(R)
         1.4.4  Send Busy Status and Extract N(R)
         1.4.5  Send Reject Condition and Extract N(R)
    1.5  Execute U-Frame Response
         1.5.1  Parse U-Frame Response Control Field
         1.5.2  Decode Response Modifier Bits
         1.5.3  Set Up Link
         1.5.4  Execute Pending State Change
    1.6  Window Primary Information Blocks
         1.6.1  Reset V(S)
         1.6.2  Update Lower Window Edge
         1.6.3  Poll Secondary for Ready State
         1.6.4  Place I-Frame in X.25 Link
         1.6.5  Select Packet for X.25 Link
    1.7  Recover from Procedure Error
    1.8  Send Packet
2.  Transmit Bit Streams
3.  Execute HDLC Protocol at Secondary Node
    3.1  Extract Valid Packet
    3.2  Decode Primary Address and Control Fields
    3.3  Execute Primary I-Frame
         3.3.1  Validate I-Frame Packet
         3.3.2  Parse I-Frame
         3.3.3  Parse I-Frame Control Field
    3.4  Execute S-Frame Command
         3.4.1  Parse S-Frame Command
         3.4.2  Parse S-Frame Control Field
         3.4.3  Send Ready Status and Extract N(R)
         3.4.4  Send Busy Status and Extract N(R)
         3.4.5  Send Reject Condition and Extract N(R)
    3.5  Execute U-Frame Command
         3.5.1  Parse U-Frame Command Control Field
         3.5.2  Decode Command Modifier Bits
         3.5.3  Disconnect Link
         3.5.4  Set Up Link
    3.6  Window Secondary Information Blocks
         3.6.1  Reset V(S)
         3.6.2  Update Lower Window Edge
         3.6.3  Set Response Final Bit
         3.6.4  Place I-Frame in X.25 Link
         3.6.5  Select Packet for X.25 Link
         3.6.6  Respond to Status Request
    3.7  Request Recovery from Procedure Error
    3.8  Send Packet
```

Protocol at Primary Node (1) process and the Execute HDLC Protocol at Secondary Node (3) process differ in that the former node must have the responsibility for resetting the channel as well as polling the status of the latter node. However, other than these additional responsibilities, the processes are very similar. The Transmit Bit Streams (2) process is performed by the physical protocol and the transmission medium and so its requirements are analyzed in a later section. First, the Execute HDLC Protocol at Primary Node (1) will be analyzed.

Execute HDLC Protocol at Primary Node. An expanded DFD of this process is shown in Figure 22. There must be an incoming bit stream from the secondary node and an outgoing bit stream to that node since HDLC is a bit-oriented protocol. This satisfies the transparency requirement that was specified to allow the transfer of binary files. The node must supply level 3 packets to the Execute HDLC Protocol at Primary Node (1) process through the routing algorithm and the interface with the host-to-host protocol. This process views the level 3 packets strictly as information blocks to be transmitted and thus the coupling between level 2 and level 3 is minimized. This allows the protocol flexibility requirement specified in Chapter II to be met much more easily. Also, transmitted information blocks from the secondary node must be able to be extracted from the incoming bit stream and supplied to the routing algorithm and host-to-host transfer mechanism.



Primary_Incoming_Bit_Stream
1.1
Invalid_Length_Secondary_Packet
Secondary_Valid_Packet
1.2 Decode Secondary Packet Address
Invalid_Secondary_Packet_Type
Secondary_to_Primary_I-Frame
1.3 Execute Secondary I-Frame Packet
Invalid_N(R)_Secondary_I-Frame
Transmitted_Secondary_Node_Level_3_Packet
I-Frame_Packet_Sequence_Info
Secondary_to_Primary_S-Frame_Response
1.4 Execute S-Frame Response
Secondary_Node_Status
Secondary_to_Primary_U-Frame_Response
1.5 Execute U-Frame Response
U-Frame_Reset_Command
1.6 Window Primary Information Blocks
Primary_Level_2_Packet
1.8 Send Packet
Primary_Outgoing_Bit_Stream
Invalid_Length_Secondary_Packet
Invalid_Secondary_Packet_Type
Invalid_Secondary_N(R)_I-Frame
1.7 Recover from Procedure Error
SARM_Command

Figure 22 Execute HDLC Protocol at Primary Node (1) DFD

The incoming bit stream must be monitored by the Extract Valid Packet (1.1) process and valid level 2 packets extracted from the stream. These level 2 packets must then be classified by their address and control fields as either information (I-frame) packets, supervisory (S-frame) response packets, or unnumbered (U-frame) response packets. The transmitted information block must be extracted from the I-frame packet and the packet sequence information in each of the types of packets must be provided to the windowing process (1.3). A windowing mechanism very similar to that used in the level 3 protocol must be used by the Execute HDLC Protocol at Primary Node (1) process to achieve flow control over the link. The Window Primary Information Blocks (1.6) process must use the packet sequence information V(S) to insure that no more than k blocks are pending acknowledgement at any time where k is the window size. The windowed information blocks as well as the command and response packet replies must be assembled into level 2 packets to be sent to the secondary node. Local procedure errors due to invalid length packets, invalid packet types, and out of sequence I-frames must be recovered from by generating a set asynchronous balanced mode (SABM) command. This command initiates the resetting of the link in both directions (1.7). Finally, the level 2 packet must be sent out of the node in the outgoing bit stream by inserting the packet between two flag characters (1.8).

<u>Extract Valid Packet</u>. The Extract Valid Packet (1.1)

process is expanded into a lower-level DFD in Figure 23. The incoming bit stream must first be monitored for the flag sequence of bits that signifies the start or end of a packet. Once this sequence is detected, all information between it and the next flag must be extracted and passed as a level 2 packet to the Unstuff Zeros (1.1.2) process. However, if an abort sequence of seven contiguous ones is detected or an idle channel state is indicated by fifteen contiguous ones, then the packet being extracted must be discarded (1.1.1). The Unstuff Zeros (1.1.2) process must remove zero bits that were stuffed in the packet to prevent the bit patterns in the packet from appearing as the flag sequence. The unstuffed packet must be checked for valid length and packets of invalid length must also be discarded (1.1.3). Finally, the valid length packet must be checked for bit errors by regenerating the cyclic redundancy check (CRC) remainder and comparing it to the frame checking sequence (FCS) in the packet. If they match, then the packet must be valid. Otherwise, the packet must be discarded (1.1.4,1.1.5).

Execute I-Frame Packet. As shown in the lower-level DFD for this process in Figure 24, the I-frame packet must first be validated by comparing the next packet expected number N(R) of the I-frame to the primary receive state variable V(R). If they do not match then the I-frame must be considered invalid (1.3.1). Otherwise, the valid I-frame packet must be parsed and the control field separated from

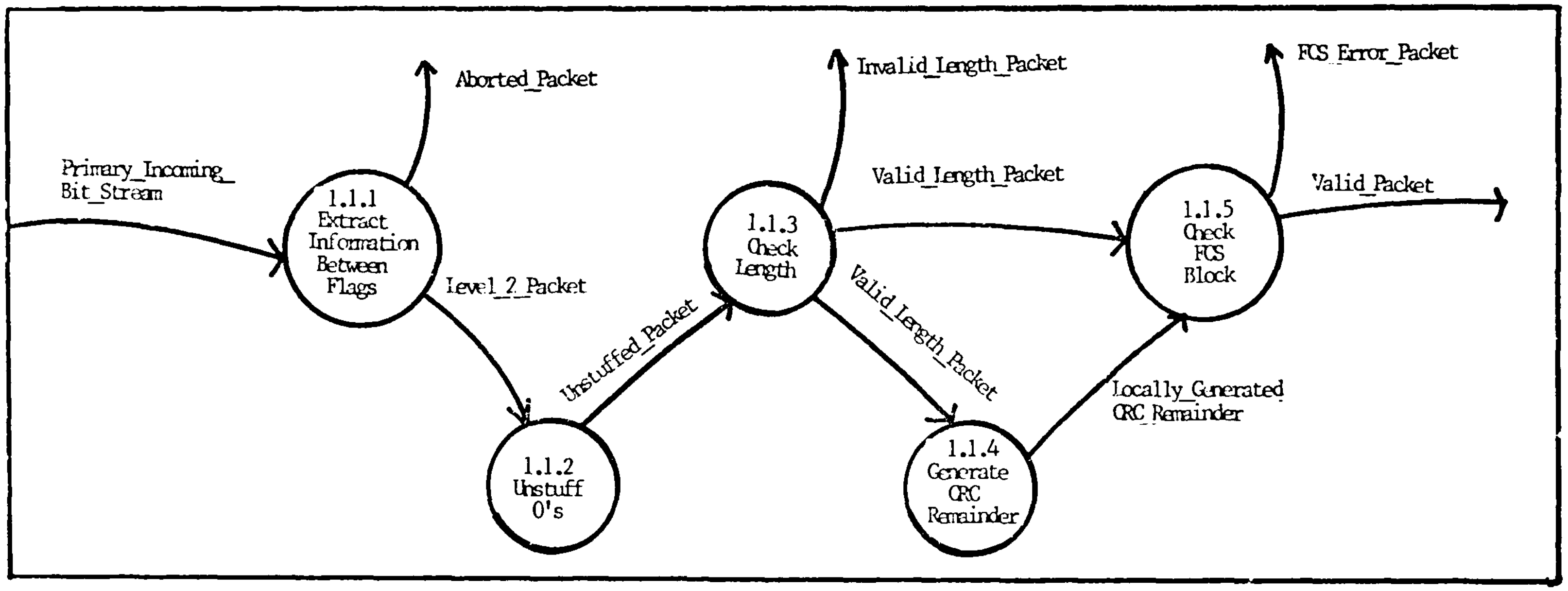


Figure 23 Extract Valid Packet (1.1) DFD

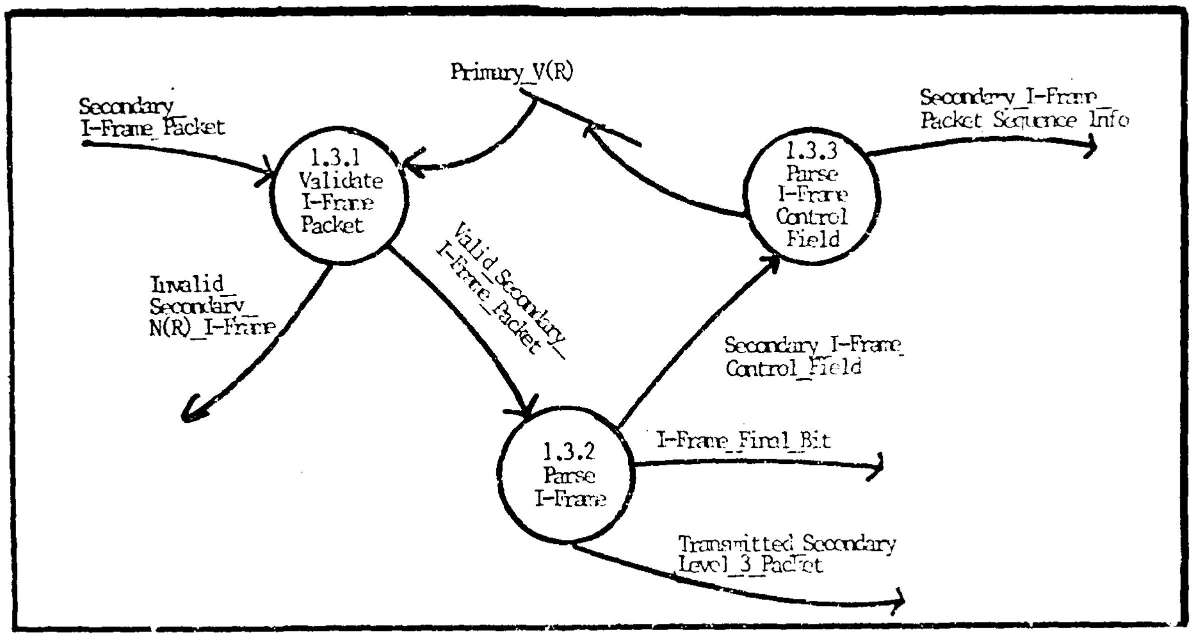

Primary_V(R)
Secondary_I-Frame_Packet
1.3.1 Validate I-Frame Packet
Secondary_I-Frame_Packet Sequence Info
1.3.3 Parse I-Frame Control Field
Valid_Secondary_I-Frame_Packet
Invalid_Secondary_N(R)_I-Frame
Secondary_I-Frame_Control_Field
I-Frame_Final_Bit
1.3.2 Parse I-Frame
Transmitted_Secondary_Level_3_Packet

Figure 24 Execute Secondary I-Frame Packet (1.3) DFD

the transmitted information block (1.3.2). The control field must then be parsed further to extract N(R) and the packet sequence number N(S). The transmitted information block must be provided to the node as a level 3 packet and N(R) and N(S) must be used to provide packet sequence information to the windowing mechanism (1.3.3).

Execute S-Frame Response. The Execute S-Frame Response (1.4) process lower-level DFD is shown in Figure 25. The S-frame must first be parsed to extract the response supervisory function bits and the final bit (1.4.1). If the final bit is set, then the response must be interpreted as the reply to the last command sent out by the process with the poll bit set. The supervisory modifier bits must be classified as either a Receive Ready (RR) response, a Receive Not Ready (RNR) response, or a Reject

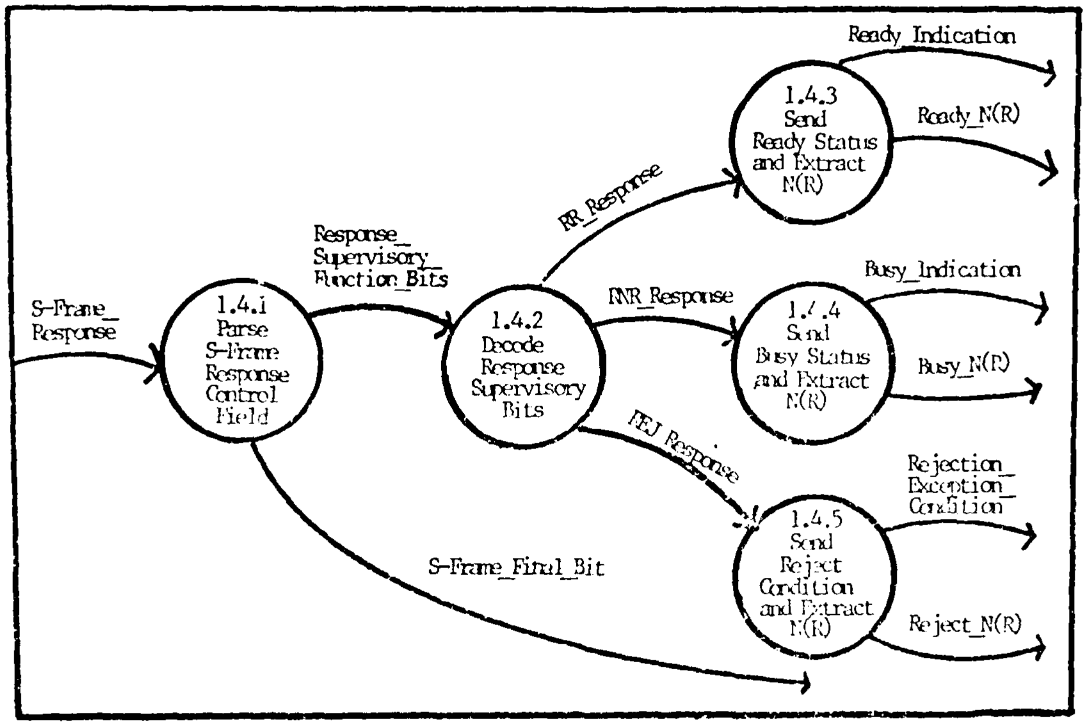

Ready_Indication
1.4.3 Send Ready Status and Extract N(R)
Ready_N(R)
RR_Response
Response_ Supervisory_ Function_Bits
S-Frame_ Response
1.4.1 Parse S-Frame Response Control Field
1.4.2 Decode Response Supervisory Bits
RNR_Response
1.4.4 Send Busy Status and Extract N(R)
Busy_Indication
Busy_N(R)
REJ_Response
1.4.5 Send Reject Condition and Extract N(R)
Rejection_ Exception_ Condition
Reject_N(R)
S-Frame_Final_Bit

Figure 25 Execute S-Frame Response (1.4) DFD

(REJ) response (1.4.2). If the S-frame response is an RR response, then the ready indication must be sent to the Window Primary Information Blocks (1 (1.4.3).6) process. Also, the N(R) of the RR response must be used to update V(R) which stores the sequence number of the next I-frame expected from the secondary node. If the S-frame response is an response, then the busy status must be sent to the windowing process and N(R) used to update V(R) (1.4.4). Finally, a REJ response must result in the N(R) of the REJ response updating V(R) and the windowing process receiving a reject exception condition (1.4.5).

<u>Execute U-Frame Response</u>. The Execute U-Frame Response (1.5) process can also be expanded further as shown

in Figure 26. The U-frame response, like the S-frame

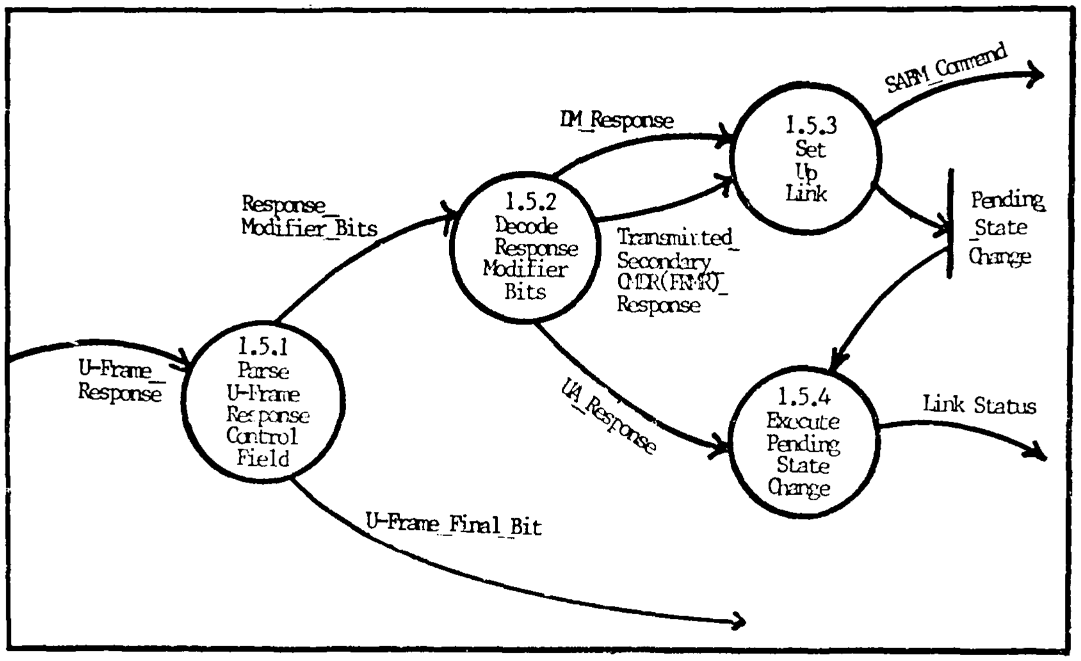

U-Frame_Response
1.5.1 Parse U-Frame Response Control Field
Response_Modifier_Bits
1.5.2 Decode Response Modifier Bits
DM_Response
Transmitted_Secondary_CMDR(FRMR)_Response
UA_Response
1.5.3 Set Up Link
SABM_Command
Pending_State Change
1.5.4 Execute Pending State Change
Link Status
U-Frame_Final_Bit

Figure 26 Execute U-Frame Response (1.5) DFD

response, must first be parsed to extract the final bit and the modifier bits (1.5.1). The final bit must be processed as it was for the S-frame response and the response modifier bits must be decoded and the U-frame response classified as either a disconnected mode (DM) response, an unnumbered (UA) response, or a command (frame) reject (CMDR(FRMR)) response (1.5.2). A DM response or a CMDR(FRMR) response from the secondary node must result in an SABM command being sent to the windowing process to initiate setting up the link between the primary and secondary nodes. Also, this pending link setup must be stored in the pending state change variable (1.5.3). If the U-frame resonse is a UA response, then the pending state change variable must be accessed to determine if a link setup or disconnect is pending. If so, the state must be changed to the pending state and the pending state change variable cleared (1.5.4).

<u>Window Primary Information Blocks</u>. As can be seen in Figure 27, this process must exercise overall control over both directions of the link transmission. A rejection exception condition must be cleared by resetting the primary send state variable V(S) to the N(R) supplied from the Execute S-Frame Response (1.4) process and by sending a reject ready indication to the Place I-Frame in X.25 Link Queue. This is done by the Reset V(S) process (1.6.1). Irregardless of whether the N(R) came from an I-frame, an RR command, an RNR command, or a REJ command, it must be used to update the lower window edge. This updated lower window edge must then be used to place I-frames in the X.25 link queue (1.6.2). The busy and ready indications must be used to place the X.25 link in the ready condition as soon as possible, if it is not in that state already (1.6.3). This is done by generating an RR command to poll the secondary node repeatedly until it indicates that it is in the ready state. When the link status input, the reject ready indication, or the ready indication indicates that the secondary node is in the ready state, then I-frames must be placed in the X.25 link queue until V(S) is one less than the updated lower window edge (modulo k) (1.6.4). Finally, the Select Packet for X.25 Link (1.6.5) process must select the level 2 packet to be sent next by servicing the I-frame

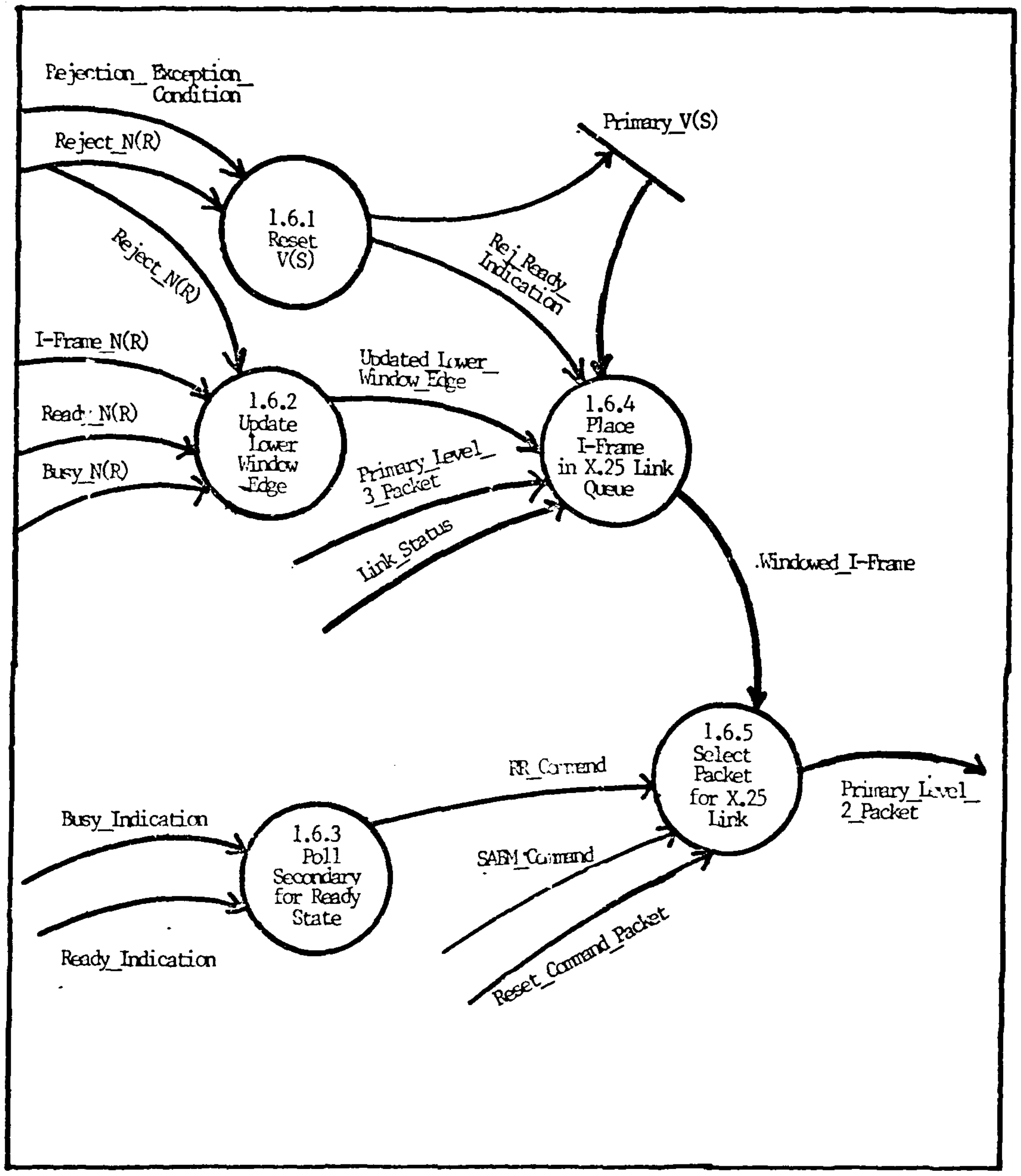


Figure 27 Window Primary Information Blocks (1.6) DFD

queue but giving priority to the SABM command, RR command, or DISC command if one is waiting to be transmitted.

Execute HDLC Protocol at Secondary Node. As can be seen from Figure 28, this process is very similar to the Execute HDLC Protocol at Primary Node (1) process. There are, however, some major differences. The valid primary packet may be either an I-frame, an S-frame command, or a U-frame command (3.2). Thus, the primary node must send S-frame and U-frame commands to the secondary node while the secondary node can only send back S-frame and U-frame responses. Also, the secondary node must request recovery from procedure errors by sending CMDR(FRMR) responses to the primary node (3.7). Finally, the primary I-frame may have a node status polling request that must be answered with either an I-frame or an S-frame response (3.3,3.4).

Execute Primary I-Frame. This process is similar to the Execute Secondary I-Frame (1.3) process but differs slightly. The lower-level DFD of the Execute Primary I-Frame (3.3) process in Figure 29 shows that the poll bit is extracted from the I-frame rather than the final bit (3.3.2). Other than this slight difference the I-frames are processed almost identically by the primary and secondary nodes.

Execute S-Frame Command. The lower-level DFD for the Execute S-Frame Command (3.4) process is shown in Figure 30. It is also identical to its counterpart with the exception of the poll bit (3.4.1).

Execute U-Frame Command. The Execute U-Frame Command (3.5) process is quite different from its counterpart in the primary node. As shown in Figure 31, the U-frame command

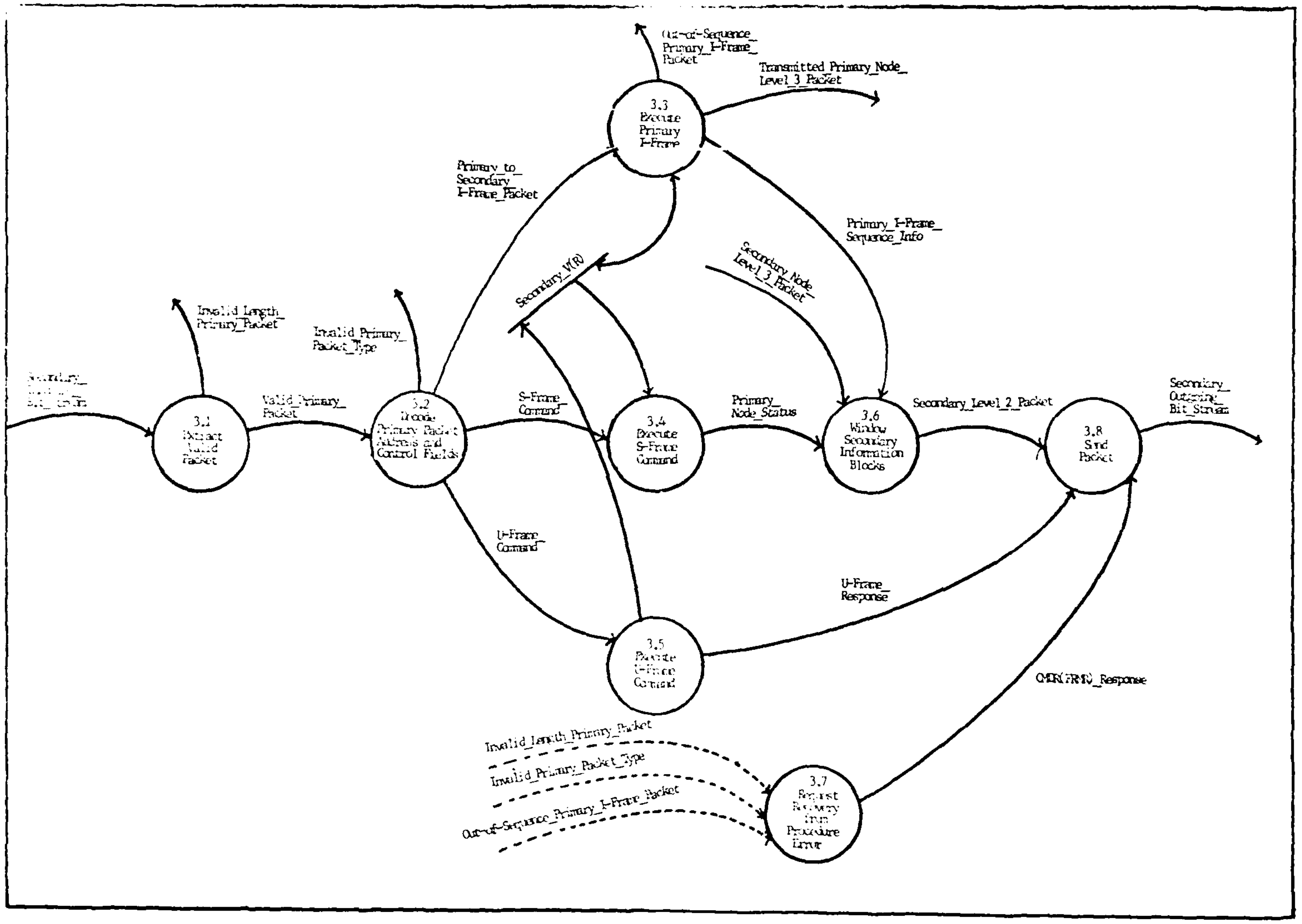

Secondary_ Incoming_ Bit_Stream
3.1 Extract Valid Packet
Invalid_Length_ Primary_Packet
Valid_Primary_ Packet
3.2 Decode Primary Packet Address and Control Fields
Invalid_Primary_ Packet_Type
Primary_to_ Secondary_ I-Frame_Packet
3.3 Execute Primary I-Frame
Out-of-Sequence_ Primary_I-Frame_ Packet
Transmitted_Primary_Node_ Level_3_Packet
Secondary_V(R)
S-Frame_ Command
3.4 Execute S-Frame Command
Primary_ Node_Status
Primary_I-Frame_ Sequence_Info
Secondary_Node_ Level_3_Packet
3.6 Window Secondary Information Blocks
Secondary_Level_2_Packet
3.8 Send Packet
Secondary_ Outgoing_ Bit_Stream
U-Frame_ Command
3.5 Execute U-Frame Command
U-Frame_ Response
CMDR(FRMR)_Response
Invalid_Length_Primary_Packet
Invalid_Primary_Packet_Type
Out-of-Sequence_Primary_I-Frame_Packet
3.7 Request Recovery from Procedure Error

Figure 28. Execute HDLC Protocol at Secondary Node (3) DFD

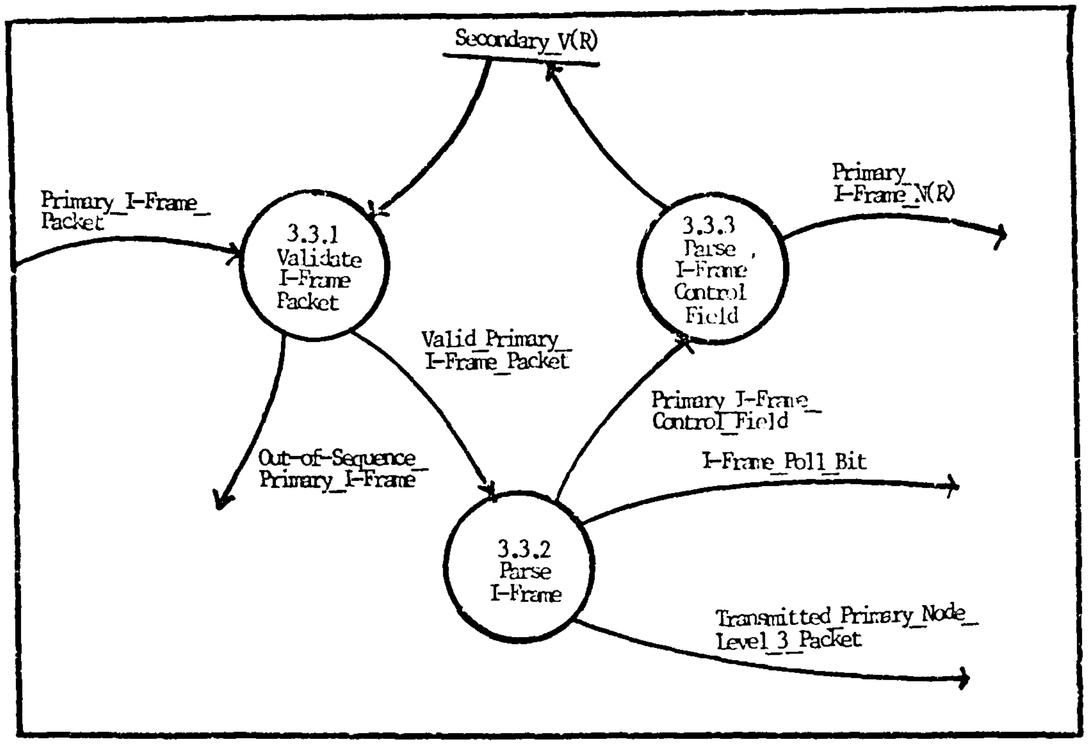

Secondary_V(R)
Primary_I-Frame_Packet
3.3.1 Validate I-Frame Packet
Valid_Primary_I-Frame_Packet
3.3.3 Parse I-Frame Control Field
Primary_I-Frame_N(R)
Primary_I-Frame_Control_Field
Out-of-Sequence_Primary_I-Frame
I-Frame_Poll_Bit
3.3.2 Parse I-Frame
Transmitted_Primary_Node_Level_3_Packet

Figure 29. Execute Primary I-Frame Packet (3.3) DFD

can be either a disconnect (DISC) command or an SABM command transmitted from the primary node (3.5.2). The DISC command must result in the secondary node entering the disconnected state and generating a UA response to be sent to the primary node. If the node was already disconnected, then a DM response must be generated instead (3.5.3). The transmitted SABM command must result in the link being reinitialized with V(R) and V(S) set to zero. Moreover, a UA response must be generated for this command (3.5.4).

<u>Window Secondary Information Blocks</u>. The lower-level DFD of this process is shown in Figure 32. In addition to the processes required in the Window Primary Information Blocks (1.6), this process also must insure that responses

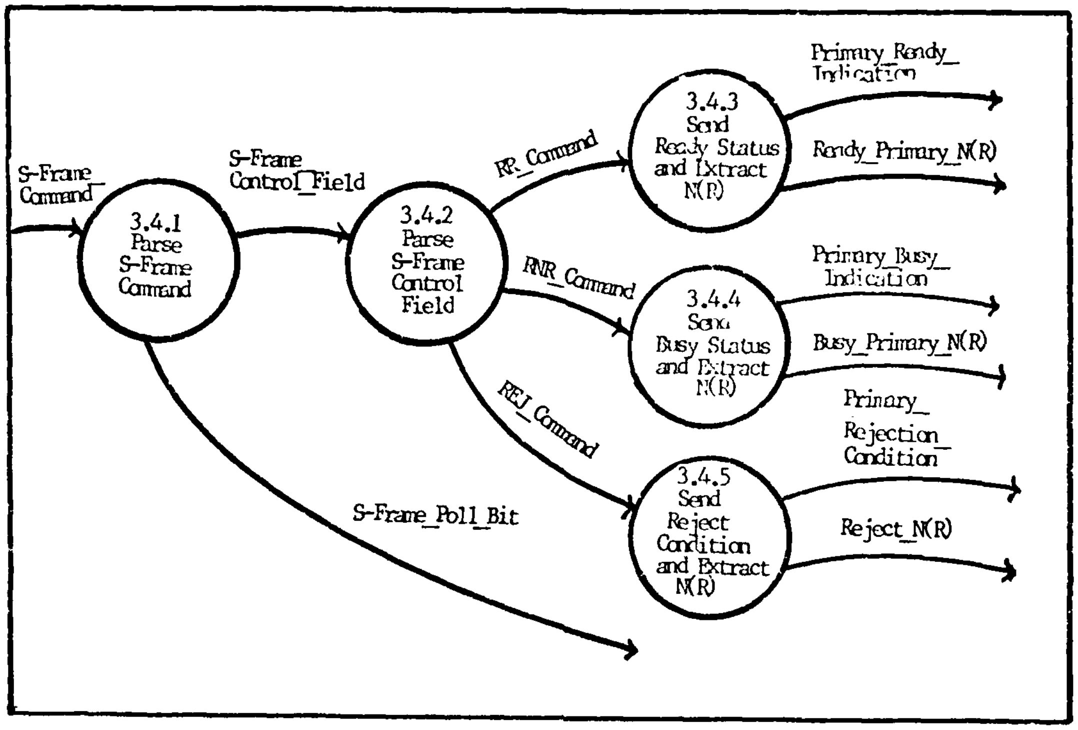

S-Frame_Command
3.4.1 Parse S-Frame Command
S-Frame Control_Field
3.4.2 Parse S-Frame Control Field
RR_Command
RNR_Command
REJ_Command
3.4.3 Send Ready Status and Extract N(R)
3.4.4 Send Busy Status and Extract N(R)
3.4.5 Send Reject Condition and Extract N(R)
Primary_Ready_Indication
Ready_Primary_N(R)
Primary_Busy_Indication
Busy_Primary_N(R)
Primary_Rejection_Condition
Reject_N(R)
S-Frame_Poll_Bit

Figure 30. Execute S-Frame Command (3.4) DFD

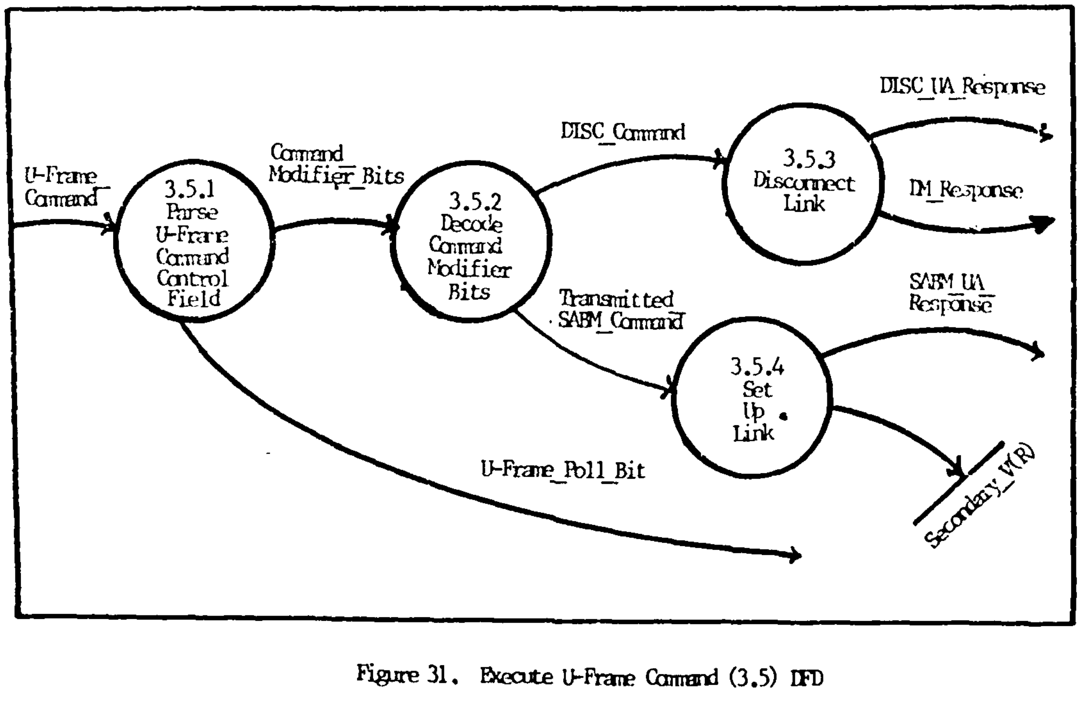


Figure 31. Execute U-Frame Command (3.5) DFD

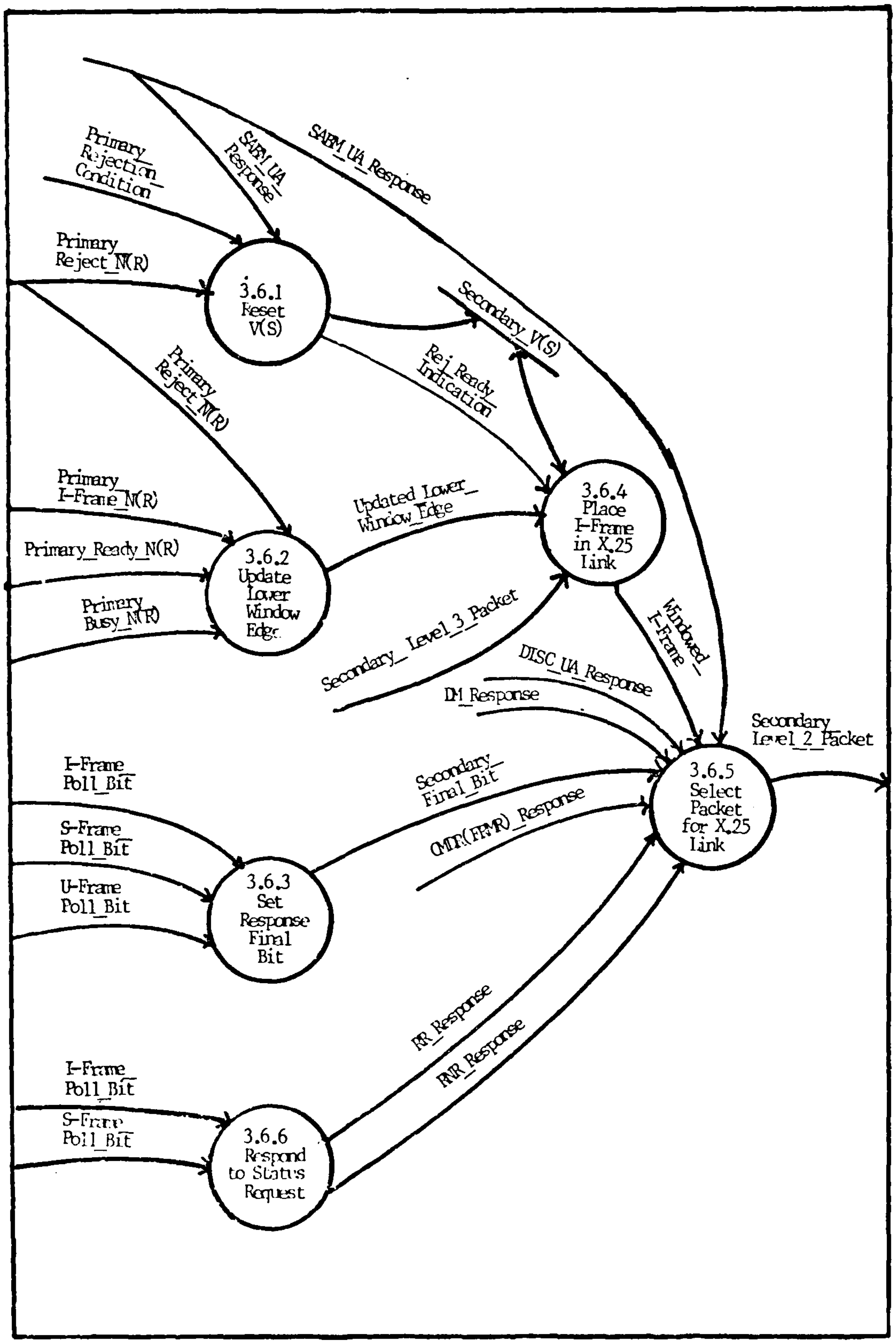


Figure 3. Window Secondary Information Blocks (3.6) DFD

are generated to all commands received from the primary node and that in those responses, the final bit is set (3.6.3,3.6.6).

Physical Protocol Requirements. The X.25 protocol requires that the RS-449 interface standard be used to implement the physical level protocol. However, levels 2 and 3 do not require that a certain physical level protocol be used. Thus, this requirement for the RS-449 interface was not considered binding. There were, then, two principal requirements for the physical level protocol. It had to be a widely recognized standard to minimize the difficulty in adding new hosts to DELNET and it had to be compatible with the transmission medium selected and the distances of transmission required. The distance from the hosts to their entry nodes should not exceed 75 feet given the present available space in the Digital Engineering Laboratory. Thus, if these requirements could be met then the Transmit Bits (2) process in the link level protocol could be performed.

Summary

This chapter has refined the hardware specifications and addressed the software specifications for DELNET in detail. The software specification was lengthy, and yet, because of the partitioning and levels of abstraction made possible through the use of Structured Analysis, it was fairly easy to comprehend. The data dictionary contains even more information on these specifications and for that reason is included as Appendix C. These requirements formed the basic foundation for the overall structural design of the DELNET software described in Chapter IV.

IV. Design of DELNET

I troduction

The previous chapters determined the user functional requirements and translated them into a detailed set of hardware requirements as well as a structured software specification. In this chapter, the hardware requirements are employed along with supporting background information to develop the hardware design. The software for DELNET is also designed using Yourdon and Constantine's structured design techniques to translate the structured specification into the module structure charts found in Appendix D. Finally, the software modules are allocated to specific processors in DELNET.

Hardware Design

Topology. Although there was a requirement that the topology be able to be easily modified, there was still a need for an initial DELNET topology. The topology shown in Figure 33 was chosen for several reasons. The basic loop architecture of the nodes allows the routing algorithm to be simple since a message has only two possible paths to its destination node. This decreases the development time of DELNET and allows it to be operational while a more sophisticated routing algorithm is being developed to handle more complex topologies. Also, the loop network has only one more link between nodes than a minimum spanning tree. This can be seen by deleting one link from the loop

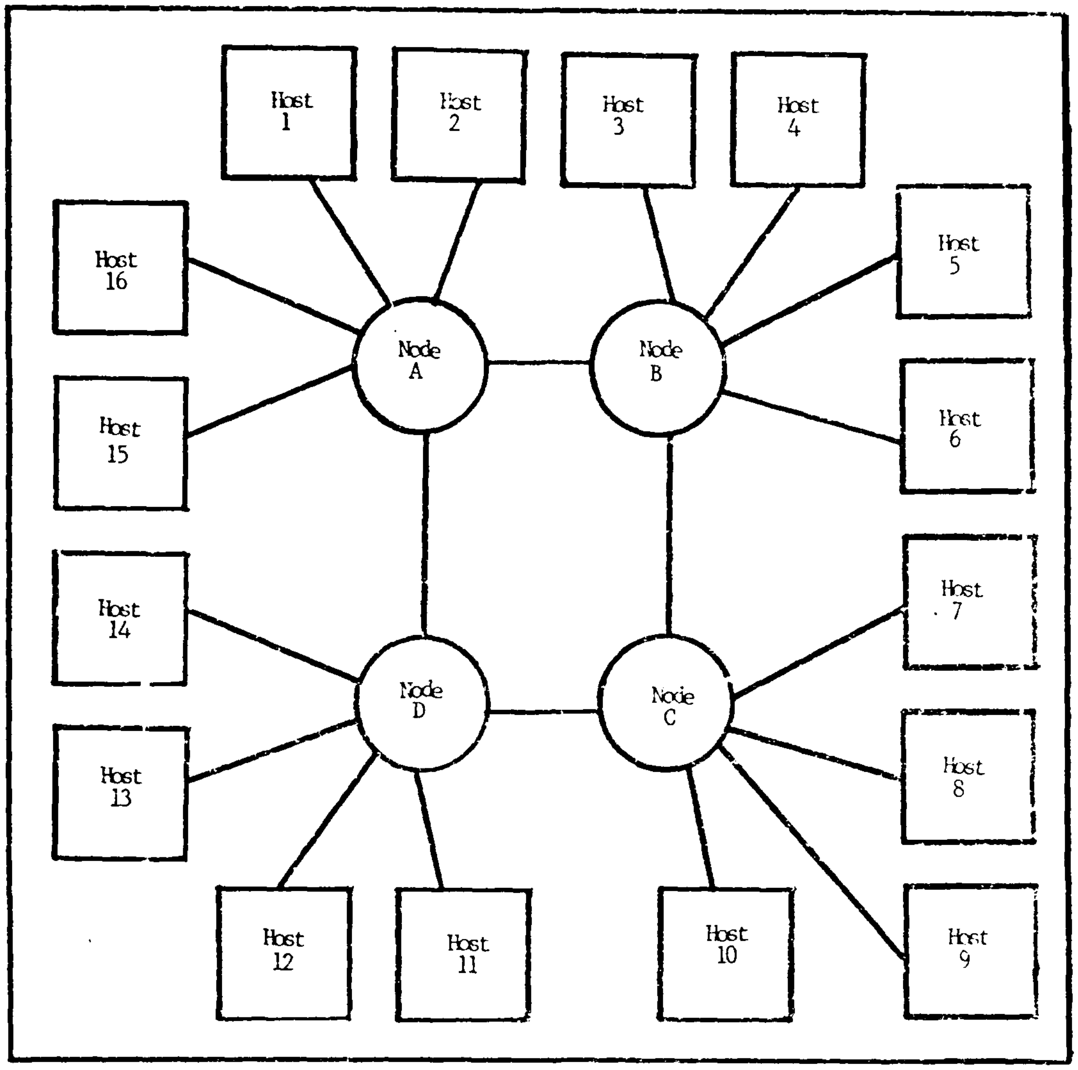


Figure 33. Basic HINET Topology

architecture. The network in this case degenerates to a linear connection. Since no more links can be removed from the retwork and leave it connected, this constitutes a minimum spanning tree. This makes the loop architecture cost-effective. Since adding another node simply requires breaking one connection and adding two new ones, the loop network is easy to expand. It is these advantages that make the loop network preferable in general for local networks and very useful in the case of DELNET in particular (Ref. 10: 75). There are disadvantages, however, to the loop topology. The reliability decreases as the number of nodes on the network increases. This is a direct consequence of the near minimum number of links in the network. Also, the response time will increase as the number of nodes increases since each node that a message must pass through adds a delay to the transit time. Since the initial number of nodes on the network will be small and since the availability was only 90 percent, these disadvantages do not pose a problem at present.

The star topology connecting the hosts to the node was chosen for three reasons. First, it minimizes the loop topology's disadvantages by decreasing the number of nodes in the network. Also, by allowing one node to interface several hosts to DELNET, it is cost-effective since the cost of the nodes can be substantial. This will be seen in a following section addressing the node design. In addition, the star topology makes it practical to interface small inexpensive computers such as the E & I Instruments Mini-Micro Designer (MMD-1) to the network (Ref. 11: Chapter IV). This is true even though the computers' costs might only be a fraction of the the node's costs. Another advantage is that the hosts that will interact the most with each other can be grouped at the same node. In this configuration, much of the traffic need only pass through

AD-A100 822 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN OF A LOCAL COMPUTER NETWORK FOR THE AIR FORCE INSTITUTE --ETC(U)
MAR 81 W C HOBART
UNCLASSIFIED AFIT/GE/EE/81M-3 NL

2 OF 4
AD A
100822

its entry node to the network. This decreases the average response time on DELNET and decreases the amount of traffic transmitted between nodes. However, reliability from the host's perspective is decreased since the failure of a node can cause several hosts to lose access to the network. Also, there must be a greater transmission rate between nodes than that between the hosts and the nodes. This is to keep the nodes from acting as bottlenecks and thereby decreasing throughput to an unacceptable level.

Hosts. Three hosts were chosen for the initial network configuration. This number represents the minimum number of hosts that would exercise the protocols sufficiently. The VAX-11/780 was chosen because it is the most powerful and sophisticated of the minicomputers in the Digital Engineering Laboratory. It also has the capability to transfers files to and from the CYBER 750 and thus helps to partially fulfill the user requirement for access to that host. The Intel Series II MDS was chosen to represent the other end of the spectrum since it is an 8080-based microcomputer and has a high rate of usage. Also, a distributed database system has been designed for this computer that should aid efforts in implementing a distributed database system on DELNET. The third host chosen was the Data General Nova. This computer has a high usage rate and is already linked through a shared disk drive to the Data General Eclipse. It is also linked to the Cyber 750 mainframe computer through a communications protocol and a 300 baud telephone line. Thus, the Nova will allow the other hosts on DELNET to have indirect access to the Cyber 750 and the computational power of the Eclipse.

Nodes. The first possibilities examined to perform the function of the nodes on DELNET were existing local network software packages that might be procured to interface the hosts selected. These were rejected for two principal reasons. The commercially available networks offered no flexibility to the protocols they employed and would have been very difficult to modify. This is especially true since these packages do not usually include the source code. The other deterrent was the high cost of these network packages. For at least one of the networks, the cost of including many of the computers in the Digital Engineering Laboratory was higher than the cost of the computers themselves.

A Universal Network Interface Device (UNID) developed by the Digital Engineering Laboratory was also studied for possible use as the node in DELNET (Refs. 4,18). The UNID offered several advantages. Since it was developed as a series of Digital Engineering Laboratory research projects, complete documentation on the development and design of the UNID is available. Thus, any required modification to the UNID's hardware would be facilitated. The UNID was developed for a multi-loop network but can be used in all topologies except a bus architecture. So, the requirement for flexibility in the topology could be satisfied. In addition, the use of two Z-80A microprocessors in each UNID allows parallel processing to take place and minimizes the time that the packets must spend being processed by the nodes. The availability of a higher level language, PL/Z, and a software development system that has been interfaced to the UNID is also a significant advantage. Finally, using the UNID in DELNET also would satisfy an existing requirement to test the UNID in an operational environment. The major disadvantage to the UNID was its fairly high cost (approximately three thousand dollars) which limits the number of UNIDs that can be procured initially for DELNET. Because the advantages of the UNID greatly outweighed the disadvantage associated with its cost, the UNID was selected as the node for DELNET.

One UNID has been built and has been tested using some basic software routines (Ref. 4). Because this UNID was still in the prototype stage, only enough parts for one additional UNID were ordered. The two UNIDs will allow eight hosts to be included in DELNET and so the three hosts in the initial configuration can be easily accomodated.

Transmission Medium. The requirement that one link in DELNET use fiber optics was met by procuring a fiber optic data link to connect the two UNIDs. This link was chosen to be implemented using fiber optics due to its requirement for a high transmission rate (56 Kbs) and its length (120 ft). The links between the UNIDs and the hosts will be implemented using twisted pair since the transmission rate is limited to 19.2 Kbaud by the serial ports on the hosts and the distances between the hosts and the UNIDs are less than 75 feet. Since no problems have been experienced in the Digital Engineering Laboratory with links of this distance, this lowest-cost option was selected. As additional UNIDs are added to DELNET, these will be interconnected using shielded twisted pair. Other mediums considered for the links included coaxial cable, broadband coax, and ribbon cable. The coaxial cable would allow higher transmission rates but was not considered justified by the throughput and response time requirements. Very high transmission rates can be obtained using broadband coax and yet the modems required for the interface to the medium are very costly. Broadband coax is, however, an attractive alternative for local networks when very high transmission rates are desired (Ref. 8). Finally, ribbon cable was rejected due to its low noise tolerance and because a bit-serial protocol was to be employed. The total hardware design is shown in Figure 34.

Software Design

Introduction. The design of the software was accomplished using structured design techniques proposed by Yourdon and Constantine. Two principal methods were used to translate the structured specification into module structure charts, transform analysis and transaction analysis. These techniques were chosen due to their direct correspondence to the DFDs specified in Chapter 3.

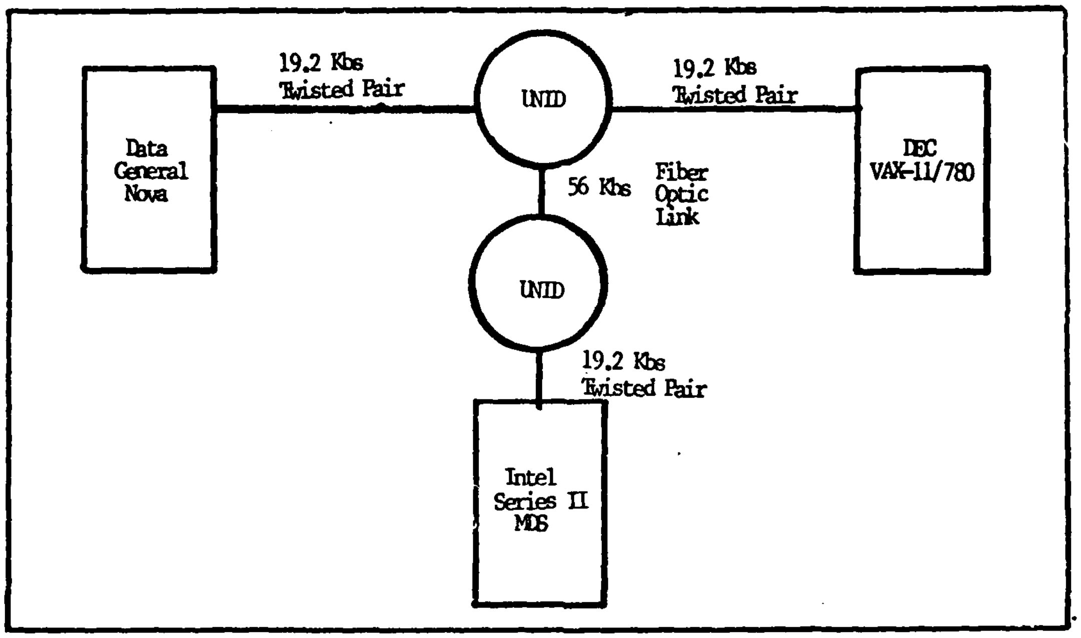


Figure 34. Initial Network Configuration

Transform analysis is used to translate a DFD into three sets of modules. The first set of modules gets data from the source. The data is transformed into the output by the second set and the third set outputs the data to the sink. To partition the DFD into the three sets of modules, the DFD is divided into an afferent branch, a transform section, and an efferent branch. This is done by tracing the input from the source to the furthest data flow where it is still recognizable as an input. Likewise, the output is traced backwards from the sink to the first data flow where it is recognizable as an output. These two data flows then are used as the boundaries between the three sections of the DFD.

For the first set of modules, the structure is derived by making an afferent module for each data flow and a transform module for each process. This is illustrated in

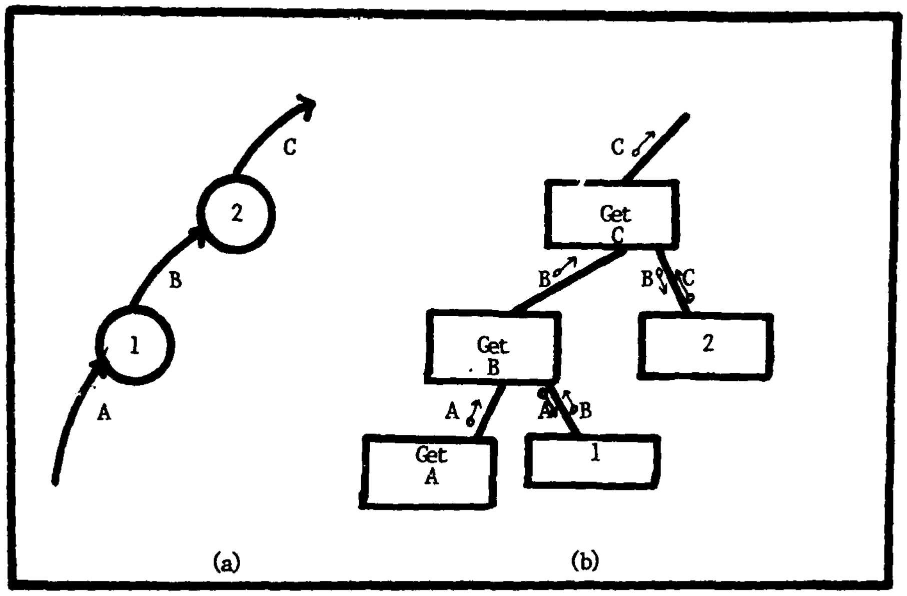


Figure 35. Factoring the Afferent Branch

Figure 35. For the second set of modules, the first module is factored into subordinate transform modules that perform the functions stated by the process names. An example of this is shown in Figure 36. The structure for the last set of modules is designed similarly to the set of afferent modules. As shown in Figure 37, the efferent module has subordinate to it another efferent module and a transform module that relates the two data flows specified by the efferent modules. A complete description of transform analysis is given in Yourdon and Constantine's book, Structured Design (Ref. 22: 187-222). The module structure chart showing all three sets of modules is shown in Figure

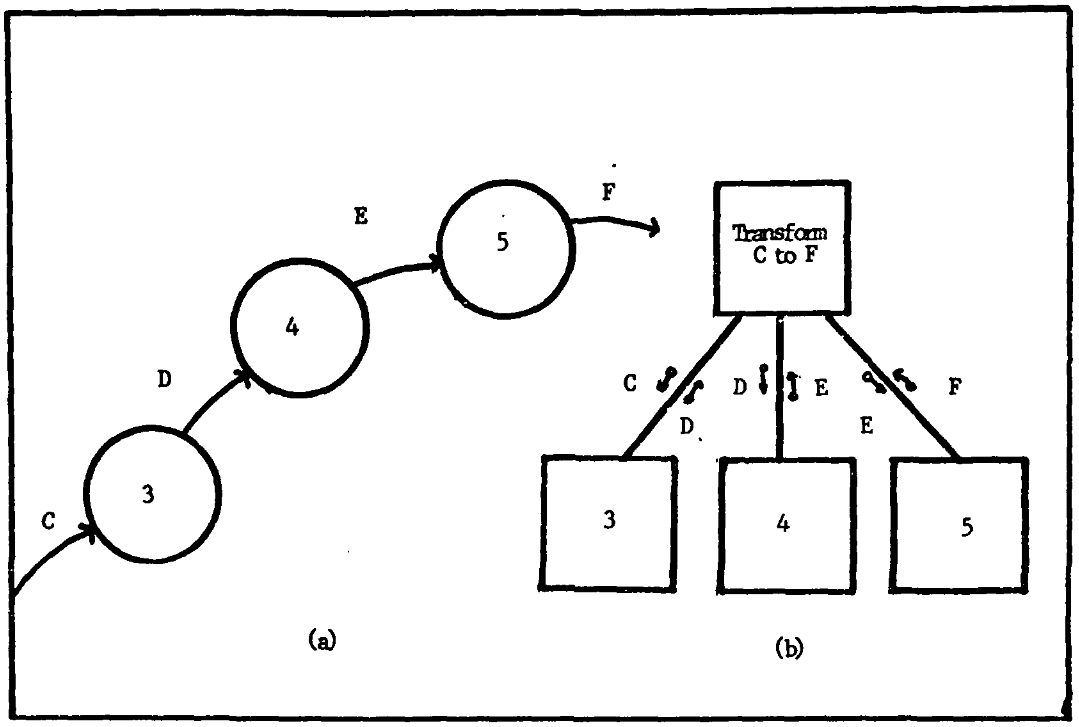


Figure 36. Factoring the Transform Section

38. This technique was used throughout the module structure design phase. An example of this technique being used in the DELNET design is shown in Figure39 for the file transfer protocol.

The other technique that was used a great deal was transaction analysis (Ref. 22: 223-229). This technique is particularly useful for translating DFDs with processes that classify an input as one of a number of outputs. Using transaction analysis, a DFD like that shown in Figure 40(a) may be translated into a module structure chart like that in Figure 40(b). The module structure chart for the help user protocol is shown in Figure 41 as an example of this technique.

Allocation of Software Modules

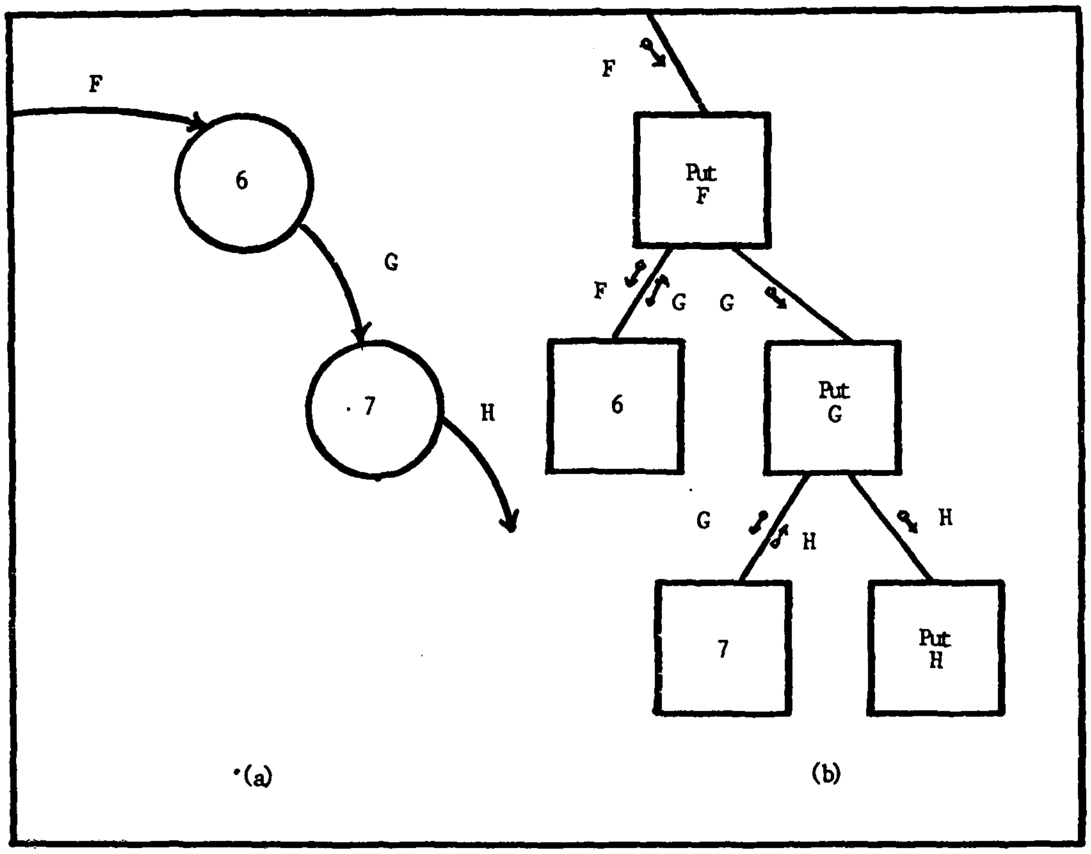


Figure 37. Factoring the Efferent Branch

A complete set of module structure charts can be found in Appendix D for the system software design. However, there are some design decisions that were made that are not obvious from the module structure charts. These decisions include the allocation of the software modules to the various processors on DELNET and also the allocation of some of the lowest level module functions to hardware devices. These allocations are discussed in the following paragraphs.

High-Level Protocol Design. The network operating system (NOS) is obviously distributed over all the processors in DELNET. However, an attempt was made to

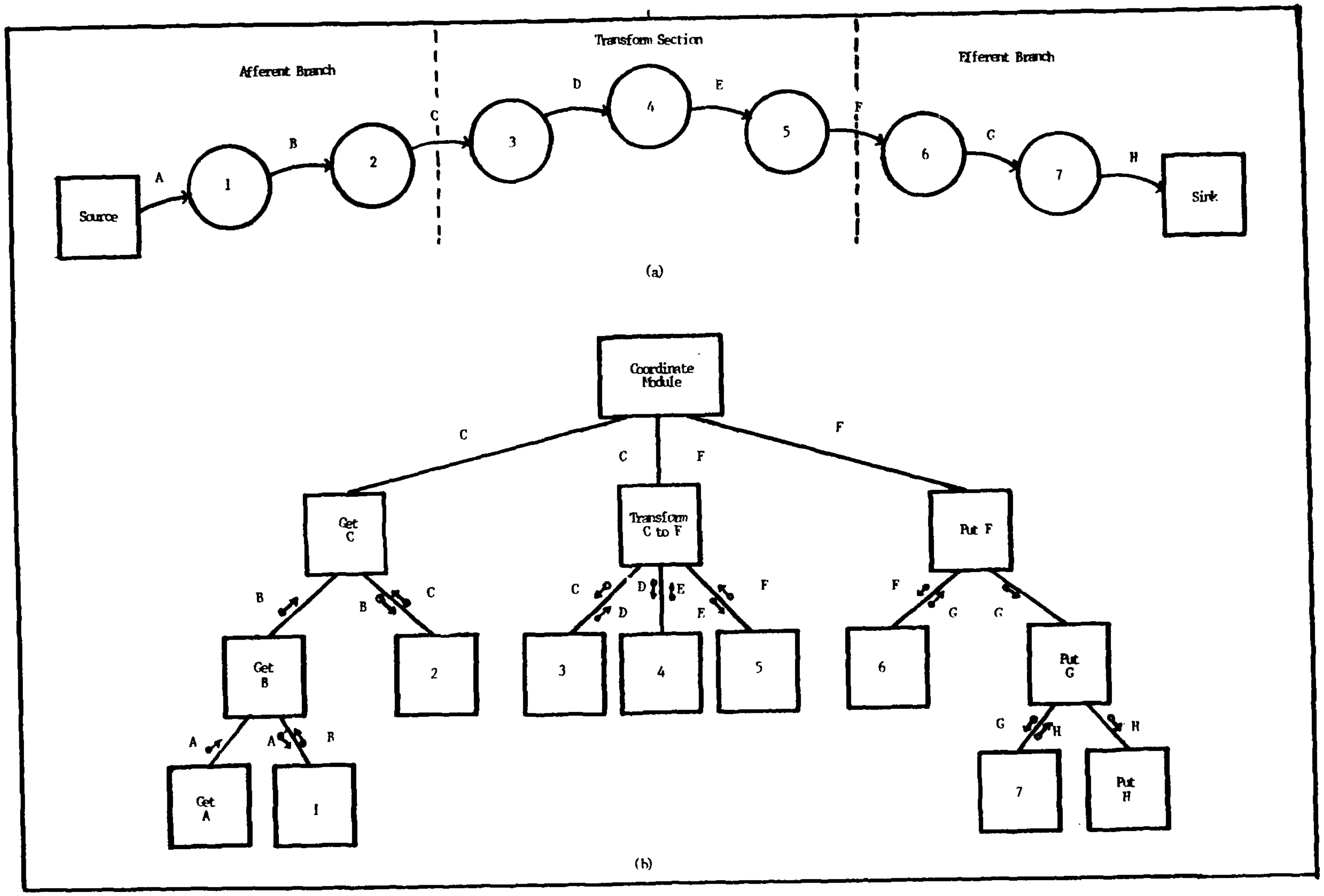


Figure 38. Transform Analysis Technique

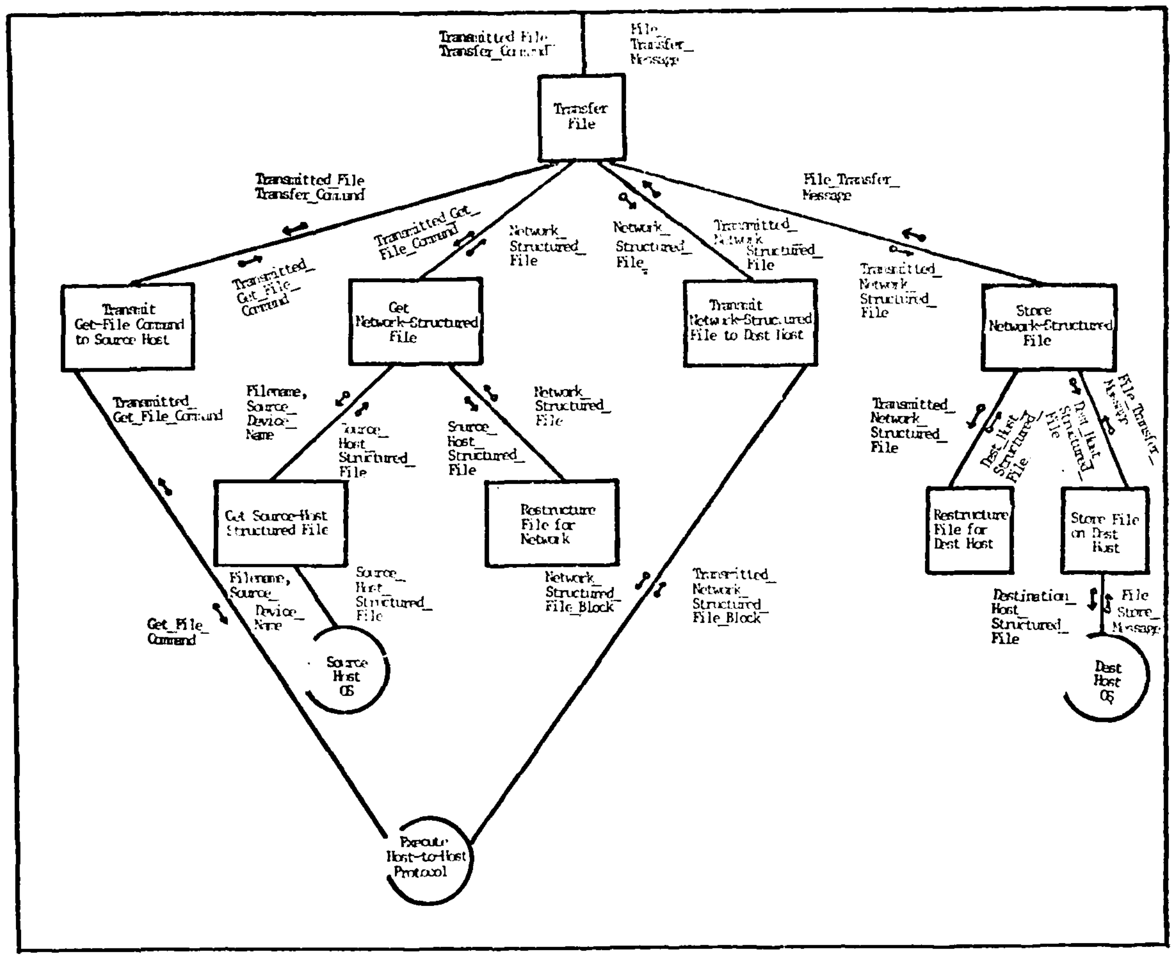

Transmitted File Transfer_Command
File_Transfer_Message
Transfer File
Transmitted File Transfer_Command
Transmitted_Get_File_Command
Transmitted Get_File_Command
Network_Structured_File
Network_Structured_File
Transmitted_Network_Structured_File
File_Transfer_Message
Transmitted_Network_Structured_File
Transmit Get-File Command to Source Host
Get Network-Structured File
Transmit Network-Structured File to Dest Host
Store Network-Structured File
Transmitted_Get_File_Command
Filename, Source_Device_Name
Source_Host_Structured_File
Source_Host_Structured_File
Network_Structured_File
Transmitted_Network_Structured_File
Dest_Host_Structured_File
Dest_Host_Structured_File
File_Transfer_Message
Get Source-Host Structured File
Restructure File for Network
Restructure File for Dest Host
Store File on Dest Host
Filename, Source_Device_Name
Source_Host_Structured_File
Network_Structured_File_Block
Transmitted_Network_Structured_File_Block
Destination_Host_Structured_File
File_Store_Message
Get_File_Command
Source Host OS
Dest Host OS
Execute Host-to-Host Protocol

Figure 39. Transfer File Module Structure Chart

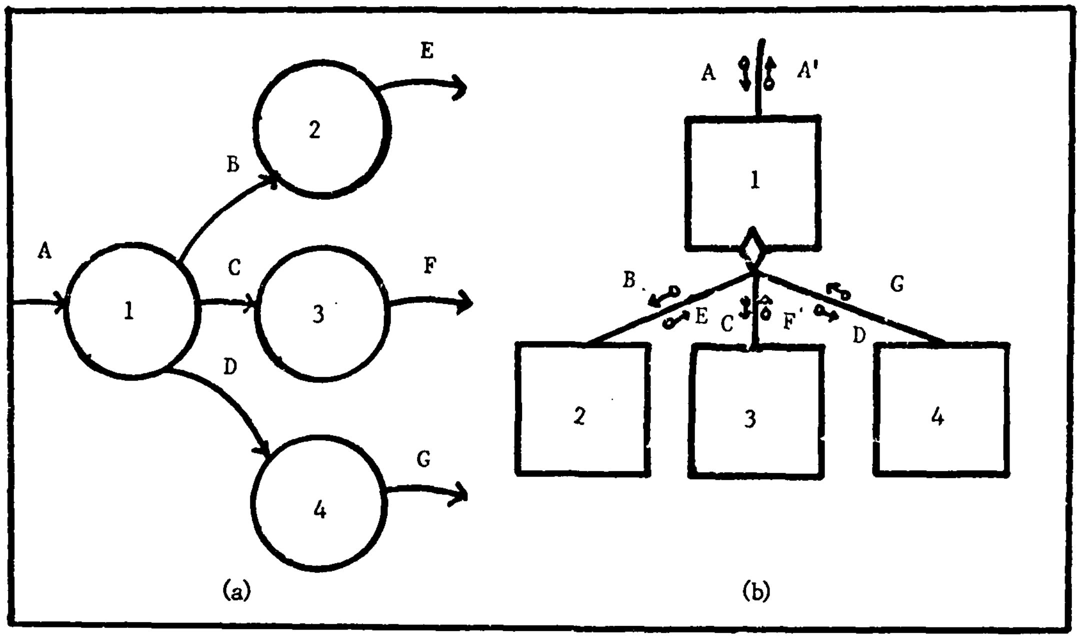
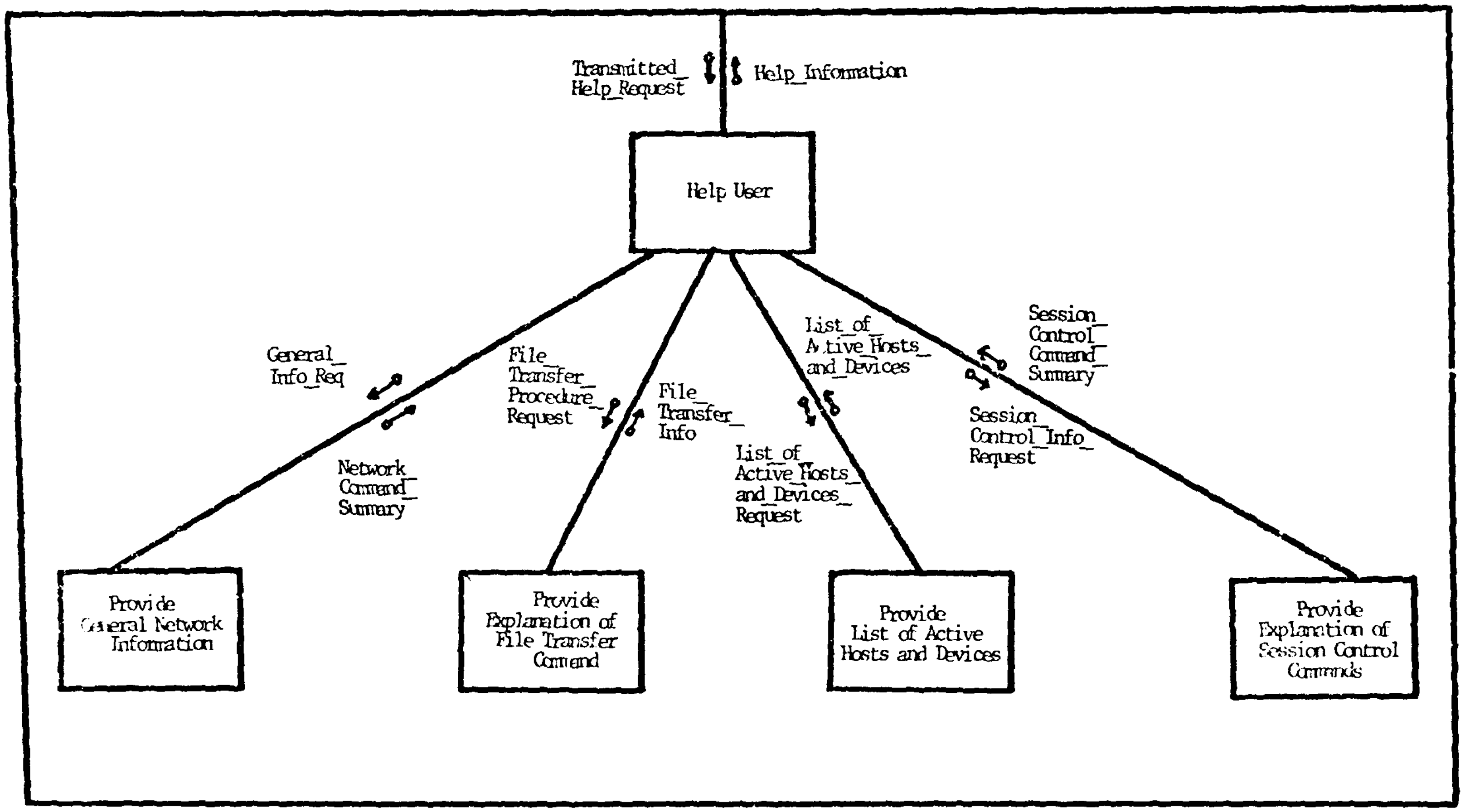


Figure 40. Transaction Analysis Technique

centralize as many functions that were not host-specific in one host for two reasons. The configuration control and ease of modification are both enhanced if only one copy of the module exists at an NOS host. If each host has a copy of each module, then a change to the network command language entails changing the software in every host. Also, the presence of modules in every host performing identical functions unnecessarily uses storage space on each host. There are some disadvantages to centralizing as much of the NOS as possible on one host, though. If the NOS host fails, then the network is not operational. Also, more traffic must be routed to the NOS host than would be the case if each host processed the network commands. Table 8 lists those modules that are present in each host and Table 9



Transmitted_Help_Request
Help_Information
Help User
General_Info_Req
Network_Command_Summary
File_Transfer_Procedure_Request
File_Transfer_Info
List_of_Active_Hosts_and_Devices
List_of_Active_Hosts_and_Devices_Request
Session_Control_Command_Summary
Session_Control_Info_Request
Provide General Network Information
Provide Explanation of File Transfer Command
Provide List of Active Hosts and Devices
Provide Explanation of Session Control Commands

Figure 41. Help User Module Structure Chart

Table 8

High-Level Protocol Modules Present at Each Host

Get User Command
Put User Response
Execute User Command
Execute Local Command
Route Local Command
Execute Routed Local Command
Route Local Response
Execute Network Command
Send Command to Host with NOS
Send Response to User
Get Network-Structured File
Get Host-Structured File
Restructure File for Network
Transmit Network-Structured File
Restructure File for Host
Store Host-Structured File
Log User into Network
Log User out of Network

lists those modules that are located only in the NOS host. Obviously, most of the modules are too host-specific to be located in the NOS host, however those modules most likely to be changed were able to be centralized there.

Lower-level Protocol Design. The availability of the

Table 9

Additional High-Level Protocol Modules
Present at NOS Host

Execute Transmitted Network Command
Transfer File
Transmit Get-File Command to Source Host
Control Session
Get List of File Transfers
Help User
Provide General Network Introduction
Provide Explanation of File Transfer Command
Provide List of Active Hosts and Devices
Provide Explanation of Session Control Commands

two processors in the UNID allowed almost all the host-to-host, network, and link level protocol modules to be allocated to the nodes. This, however, generated an additional requirement for the host-to-node interface. A virtual terminal protocol was required to make the node appear as a terminal to its hosts. Thus, the overall allocation of modules between the processors was as shown in Figure 42 .

By locating the calling host protocol in the local side of the UNID, the amount of work necessary to implement this protocol is decreased significantly. This is because the modules are not host-specific and need only be written

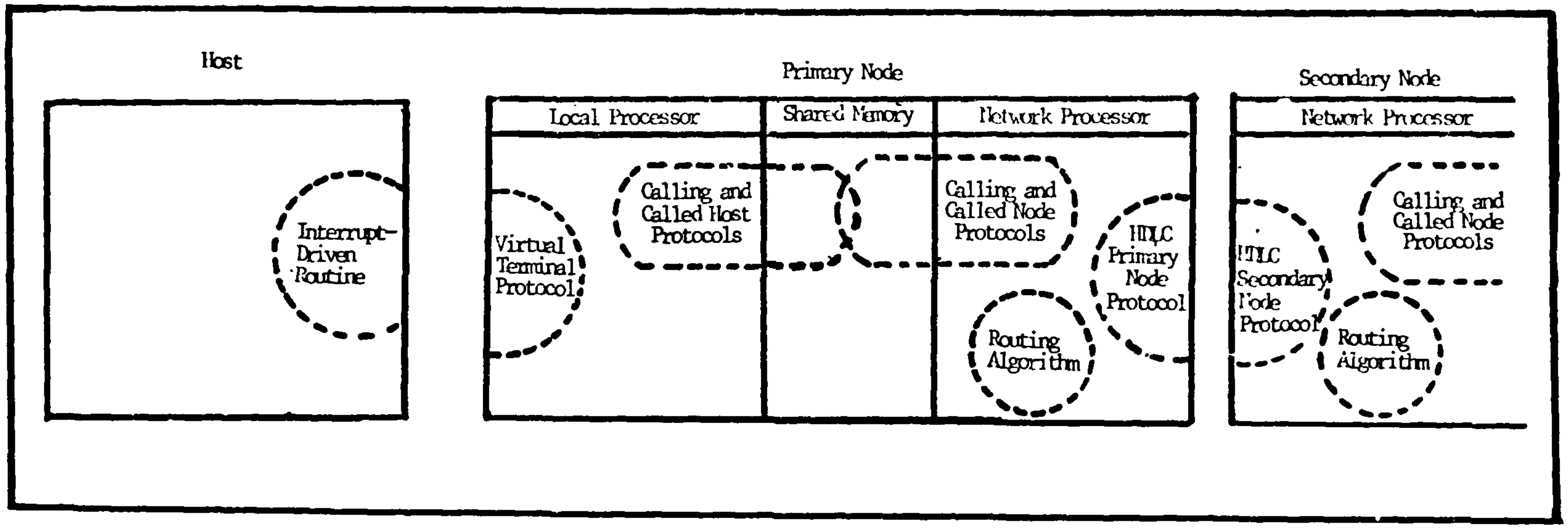

Host
Interrupt-Driven Routine
Primary Node
Local Processor
Shared Memory
Network Processor
Virtual Terminal Protocol
Calling and Called Host Protocols
Calling and Called Node Protocols
HDLC Primary Node Protocol
Routing Algorithm
Secondary Node
Network Processor
HDLC Secondary Node Protocol
Calling and Called Node Protocols
Routing Algorithm

Figure 42. Allocation of Lower-level Software Modules

once. The virtual terminal protocol is the only host-specific interfacing requirement and this can be implemented with an interrupt-driven routine on single-user hosts and by sending the necessary operating system commands on a multi-user system like the VAX-11/780 to create a file from a terminal and to output a file to a terminal.

Another advantage to allocating the calling host protocol to the local side of the UNID is that the calling host and the calling node protocols can communicate through 32 Kbytes of shared memory in the UNID. There are also 32 Kbytes of memory for each processor that enables all buffering and packet assembly to be done in the node. The shared memory is much more reliable and so this is also an advantage.

The same advantages hold for allocating the called host protocol also to the local side of the UNID. In addition, because the calling host and called host protocols are so similar, most of the modules can be shared by both. Likewise, the calling node and called node protocols can also share most modules between them.

The routing algorithm module must be allocated to the network side of the UNID as must be the link protocol modules. However, almost all of the Extract Valid Packet Module and Insert Packet in the Bit Stream Module can be accomplished by the Z-80 Serial Input Output (SIO) integrated circuit. This SIO is programmable and can accomplish many of the functions such as CRC checking at a much faster rate than could be done by the network processor (Ref. 23).

Summary

This chapter has translated the hardware and software requirements into a system design. The hardware design specifies the topology to be used, the hosts to be included, the specific node to use, and the transmission mediums to be used for each of the links. Each of these design decisions is based upon the requirements analysis and supported by background data gained from researching the various design options. The software design is based upon the structured specification and was derived primarily using transform analysis and transaction analysis. These techniques are very briefly described and the module structure charts are included in Appendix D. Finally, the software modules were allocated to the processors in the network and this allocation design was supported by an analysis of the impact of each allocation decision.

V. Implementation and Testing of the Network Command Language Interpreter

Introduction

The modules in the NOS host that interpret the network commands entered by the user have been implemented. In addition, the "help" user protocol was implemented completely. The unique factors that impacted this implementation are discussed in this chapter as are the basic principles used in implementing the modules. Also, the testing procedures that were employed to verify the modules are summarized. Finally, the testing results are stated.

Implementation

One of the first challenges in the implementation phase was to devise a log-in procedure to the network that would not be host-specific. The word, "NETWORK," was chosen as the character string to be used for two reasons. It does not contain any special characters that might be misinterpreted by the host's operating system. Also, this word is easy for the user to remember and so it improves the man-machine interface. This module was implemented on the VAX-11/780 by placing a symbol assignment in the operating system routine that logs in the user. This symbol assignment equates the character string "NETWORK" to a command procedure file call. Thus, typing in "NETWORK" causes a command procedure file to be executed which then activates the main network module that interprets the network commands. Although it was possible to implement this module on the VAX-11/780 without modifying the actual operating system, this may not be possible on the Nova and Intel Series II.

Another man-machine interface consideration was the structure of the network commands. These commands were implemented with position-independent parameters. This greatly simplifies for the user the task of learning the network command structure and also decreases the number of erroneous commands that will be entered (Ref. 13: 11-15). Instead, the short abbreviations, FN (filename), DD (destination device), DH (destination host), SD (source device), and SH (source host) are used to identify the parameters. Moreover, if any parameter is entered erroneously, then the user is prompted for a correct entry for that parameter. Missing parameters are also handled in this manner.

The user has the option of entering either the full keywords for the commands or just the first letters of the keywords. This keeps a regular user of the system from being hampered by long commands and yet keeps a novice or infrequent user from having to cope with cryptic one-letter commands and parameters. The list of keywords in the network command language is given in Table 10.

The user "help" protocol was implemented and can provide the user with all information required to use

Table 10

Valid Keywords

Network Command Parameter

LOGIN
LOGOUT
TRANSFER FILE
HELP

File Transfer Parameters

Source Host (SH=), Destination Host (DH=),

INTEL
NOVA
VAX

Source Device (SD=)

FLOPPYDISK
HARDDISK
TAPE

Destination Device (DD=)

CONSOLE
FLOPPYDISK
HARDDISK
PRINTER
TAPE

Local Host Parameter

Null (Defaults to user host)
INTEL
NOVA
VAX

Help Parameter

FILE TRANSFER
LIST CONFIGURATION
SESSION COMMANDS

DELNET. This protocol is tiered so that entry into to the protocol can be accomplished by simply typing "HELP" and yet with the addition of another keyword more specific information can be gained.

One factor that significantly increases the implementation effort is the need for complete error detection and recovery. Every module had to be protected to withstand any user input irregardless of its likelihood. This is because a process in a remote host aborting due to an out-of-range variable or other invalid input would never be perceived by the user who input the command. Thus, error recovery would be almost impossible for that user. Therefore, every module was implemented to validate the inputs first and issue error messages to be transmitted back to the user in lieu of the normal response if an invalid input was detected. This greatly increased the size and complexity of the modules and yet did succeed in making the modules impervious to invalid user inputs.

During implementation, every attempt was made to continue the practices employed during the requirements specification and design. In particular, each module was implemented such that it hid the data structures and algorithms employed as much as possible from other modules. One example of this was in the Transfer File module. This module was passed the string of characters that followed the transfer file keyword. Thus, the superordinate module did not require any knowledge of the actual number of file transfer parameters. Data coupling between modules was also an objective as was functional cohesion within the modules. These objectives were satisfied where possible in the design phase and yet care had to be taken in the implementation phase to insure that the coupling and cohesion previously achieved were not compromised.

Finally, careful consideration was given to the data structures to be employed. For example, the tradeoffs of using a linked list versus an array to store the character string representing the network command were thoroughly studied before the array was chosen. The array offered faster direct access to key positions in each parameter field and was simpler to implement. On the other hand, the linked list allowed parameter lengths and command lengths to vary more freely and would require less storage space than the array. This would be especially true if the export mode was used most often. The requirements for ease of maintenance and the desire to limit command lengths to one packet (128 bytes) made the array more attractive.

The last phase of implementation for each module consisted of debugging the module. Care was taken to insure that changes made in this phase did not destroy the structure of the module originally implemented. The code for these modules is included in Appendix E. Once the modules were free of compilation errors, they were tested using stubs and drivers as well as the techniques described in the following section.

Testing

Because of the need for complete error detection and recovery, the testing had to be rigorous. As a starting point, inputs were chosen to insure that all segments of code were executed. This was done by choosing inputs to cause each branch to be taken. Also, when a branch depended upon a compound logical expression, inputs were chosen for each of the possible logical combinations.

Equivalence class testing was also used. This procedure entailed breaking the input into its components and testing combinations of valid components until all equivalence classes of valid components had been covered. Then invalid components from each equivalence class were tested individually. As an example of this method, the testing of the transfer file command is described. First, valid file transfer parameters were used in the file transfer commands until all valid keywords had been included in at least one command. Then, an invalid file transfer parameter was tested with all other file transfer parameters being correct. This second procedure was repeated until one invalid keyword had been tested for each of the file transfer parameters.

Boundary value analysis was also used to test the modules. The values at the borders of the equivalence classes were tested to detect "off by one" errors. This proved to be one of the most useful techniques. As an example of this technique, the testing of the network command parameter is described. This parameter could not exceed twenty characters. To test the behavior of the module that parsed the network command, network command parameters that were zero, one, nineteen, twenty, and twenty-one characters long were used.

The testing was done incrementally. As a module was coded and debugged, it was then integrated into the set of modules that had already been tested. This new set of modules was then rigorously tested using the techniques already mentioned. This procedure was repeated until the test of the last module was actually a test of the complete network command language interpreter. This method worked very well and allowed interface problems between modules to be isolated quickly. Finally, records were kept of each of these tests to aid in maintaining the program. Using these documented results, a modification to the network command language can be verified by using the same input data in addition to new data to test the extensions or modifications to the network command language. These test results are included as Appendix F.

The goal of the testing was more than to just give some assurance that the program executed correctly. It was also to characterize the response of the program to the class of possible inputs. While this is not an attainable goal, due to the large universe of inputs that are possible, it did succeed in forcing the testing effort to exceed that required to convince the author of the program that his program was correct. After testing was completed, there were no known inputs that resulted in a logical error in the program.

Summary

The network command language interpreter and the user help protocol were both implemented. The network log in procedure was implemented such that it was not host-specific. Also, the commands were implemented with position-independent parameters and made interactive so that the user could be prompted for invalid inputs. Both a novice and expert mode were made possible by the implementation and the modules were implemented to validate all inputs. Both information hiding and a careful selection of data structures were employed during the implementation. Finally, the modules were tested incrementally using path analysis, equivalence class testing, and boundary value analysis.

VI. Conclusions and Recommendations

From the outset of this investigation, the development of DELNET was based upon the actual requirements of the Digital Engineering Laboratory. The user interviews were employed both to determine these requirements and to document them. From these requirements the functional requirements of DELNET were determined.

The functional requirements were used to determine the hardware specifications and the software structured specification. This portion of the investigation was the most time-consuming. In determining these specifications, however, design decisions had to be made. Thus, the process was iterative.

The time invested in the structured specification did prove worthwhile, though, when the design phase was started. Because of the amount of partitioning that had already been done, the design proved to be fairly straightforward. Structured design techniques were employed and then the software modules were allocated to the processors in the network.

The software modules were also implemented using structured techniques. Module coupling and cohesion were monitored and structured programming was employed. The most difficulty was encountered in the interface to the VAX-11/780 operating system to implement the "Get User Command" module on that host.

The testing was conducted thoroughly using branch analysis, equivalence class testing, and boundary value analysis. The implemented modules were also tested incrementally which simplified the overall system testing greatly.

In summary, through the user interviews, Structured Analysis techniques, and Structured Design techniques, a top-down design of DELNET was achieved. All primary requirements of the Digital Engineering Laboratory for file transfer capability and resource sharing have been met in the design as have the pedagogical requirements for flexibility in the topology, protocols, and transmission medium. The structured approach allowed the interfaces between the layers of protocol to be clearly defined and together the structured specification and the module structure charts will allow the implementation of DELNET by the follow-on investigations.

Recommendations

Because of the structured approach, it is possible for the implementation to be accomplished through concurrent research projects. One of the research projects should be directed towards interfacing the host to the UNID and implementing the rest of the high-level modules. The specific tasks involved in this effort are listed below:

1. Implement virtual terminal protocol for UNID
2. ment file restructuring modules for each host

3. Implement get-file, store-file, and user-command modules for each host

4. Implement command routing table in each host

5. Test network and gather performance data

6. Update network users manual

7. If time permits, interface Nova to DELNET through the Cromemco rather than through its serial port

8. If time permits, add more hosts to the network

The other research project should require that the X.25 modules and the routing algorithm module be implemented in the UNID. These objectives will also require a thorough understanding of the UNID. The specific tasks for this investigation are listed below:

1. Implement modules listed on module structure charts for Level 3

   a. Suggest using Zilog software development station and PL/Z and then using code generator to generate Z-80 code for the UNID

   b. Test and debug network protocols

2. Implement modules listed on module structure chart for Level 2

   a. Some modules can be implemented in PL/Z, others are performed by the hardware, others must be written in assembly language

   b. Implement physical links and test (includes a fiber optic link)

c. Test and debug link protocols

3. Implement the routing algorithm module and its lookup table in the UNID.

4. Collect performance data on X.25 protocol (eg., efficiency, throughput, response times)

With these two follow-on investigations, DELNET should have an initial operational capability. Additional follow-on efforts to enhance the network are also recommended, however. Because there was some interest in a distributed database on DELNET, one should be implemented. A design has been formulated for this database in another research project at the Air Force Institute of Technology (Ref. 17). Also, a distributed processing layer should be developed eventually to enable the sharing of software tools. More UNIDs should be procured and more hosts should be added to DELNET to enhance its usefulness as a tool for research in local computer networks. Implementing other protocols on DELNET will also be necessary if the network is to be used for research in this area. In addition, a more complex routing algorithm may be required as the number of nodes increases and will definitely be required for more complex topologies that might later be used. Thus, it is recommended that the routing algorithm recommended by Ravenscroft also be implemented (Ref. 16). Finally, the usefulness of the network will be enhanced by the addition of remote bootstrap capability and gateways to the CYBER 750, the ARPANET, and AFITNET.

Bibliography

1. Adams, R. Cade. A Distributed Mini-Computer Network. MS Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1977. (AD-A055-254)

2. Alford, M.W. and I.F. Burns. "R-Nets: A Graph Model for Real-Time Software Requirements," Proceedings of the Symposium on Computer Software Engineering. New York: Polytechnic Press, 1976.

3. Atlantic Research Corporation. Technical Presentation X.25-SDLC Tutorial. Atlantic Research Corporation, 1980.

4. Baker, Lee R. Prototype and Software Development for Universal Network Interface Device. MS Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1980.

5. Bass, Charlie et al. "Local Network Gives New Flexibility to Distributed Processing," Electronics, 53: 114-122 (September 25, 1980).

6. DeMarco, Tom. Structured Analysis and System Specification. New York: Yourdon, Inc., 1979.

7. Digital Equipment Corporation. Distributed Processing and Networks, A Technical Overview of Digital's Networking Products and Capabilities. Manufacturer's data. Maynard, Mass.: Digital Equipment Corp., 1980.

8. Dineson, Mark A. and J.J. Picazo. "Broadband Technology Magnifies Local Networking Capability," Data Communications, 9: 61-79 (February 1980).

9. Folts, Harold C. and Harry R. Carp (Editors). Data Communications Standards. New York: McGraw Hill Publications Co., 1978.

10. IEEE Press. Conference on Local Computer Networks. New York: IEEE Press, Inc., 1979.

11. Larsen, David G. et al. The Bugbook VI. Derby, Connecticut: E & L Instruments, Inc., 1977.

12. Liebowitz, Burt H. and John H. Carson. Tutorial Distributed Processing-- COMPCON Fall 77. New York: IEEE Press, 1977.

13. Martin, James. Design of Man-Computer Dialogues. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1973.

14. Metcalfe, Robert M. and David R. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks. Technical Report CSL-75. Xerox, Inc., Palo Alto, California. May 7, 1975.

15. Myers, Glenford J. Composite/Structured Design. New York: Van Nostrand Reinhold Co., 1978.

16. Ravenscroft, Donald L. Electrical Engineering Digital Design Laboratory Communications Network, Parts 1 and 2. MS Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1978. (AD-A064-729)

17. Roth, Mark A. The Design and Implementation of a Pedagogical Relational Database System. MS Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1979. (AD-A080-395)

18. Sluzevich, Sam C. Preliminary Design of a Universal Network Interface Device. MS Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1978. (AD-A064-059)

19. Softech, Inc. An Introduction to SADT--Structured Analysis and Design Technique. Technical Report 9022-78R. Softech, Inc. Waltham, Massachusetts. November, 1976.

20. Thurber, Kenneth J. Tutorial: Distributed Processor Communication Architecture. New York: IEEE Press, 1979.

21. Thurber, Kenneth J. and Harvey A. Freeman. "Architecture Considerations for Local Computer Networks," Proceedings, 1st International Conference on Distributed Computer Systems. 131-141. New York: IEEE Press, October 1979.

22. Yourdon, Edward and Larry L. Constantine. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Englewood Cliffs, N.J.: Prentice Hall, Inc., 1978.

23. Zilog, Inc. Z80-SIO Product Specification, Preliminary. Manufacturer's data. Cupertino, Calif.: Zilog, Inc., March 1978.

Appendix A

User Interview Results

This appendix contains a compilation of the results of the user interviews that were conducted in the requirements analysis phase of the investigation. These results provided the basis for the specification of the DELNET functional requirements in Chapter II.

Interview Outline

Introductory Narrative

I am in the preliminary design phase for the Digital Engineering Laboratory Local Computer Network (DELNET) and am attempting to determine the uses envisioned for DELNET as well as all functional requirements associated with it. For this reason, I would appreciate any help that you can give me in accurately determining these requirements. Hopefully, the questions that I have prepared in the interview outline will provide a framework in which you can communicate your ideas to me in this area.

The interview is divided into three sections. The first section lists typical uses of local computer networks, asks you to evaluate the benefits of having the capability for each use on DELNET, and then asks you to specify which uses you would like to see implemented first. The second section lists some of the functional requirements that must be determined and asks specific questions dealing with each of these requirements. At the end of this section, you will be asked to rate each of the functional requirements on a scale from not applicable to essential. Finally, the third section requests that you express any ideas that you have concerning DELNET that were not expressed by your responses to the questions in the first two sections.

Name of the Person Interviewed______________________________

Date of Interview__________________

Courses Taught by Professor__________________________________

Section I Projected Uses of DELNET

A. Resource Sharing

1. Peripheral Sharing--Capability to access network from any terminal and access any peripheral in the network from any host.

2. File Access and Transfer--Capability to transfer files between the devices and the hosts with all file restructuring transparent to the user.

3. Software Tool Sharing--Capability to access programs, compilers, cross-assemblers, and utility routines anywhere in the network for the user's program.

4. Access to ARPANET--Capability to access ARPANET from any terminal.

5. Access to CYBER 750--Capability to access CYBER from any terminal.

6. Access to AFITNET--Capability to access AFITNET from any terminal. AFITNET is a network of computers that will execute much of the AFIT workload that is currently on the CYBER.

B. Distributed Processing--Capability of executing job processes that can run concurrently on different computers.

C. Distributed Databases--Capability to access and maintain databases that are distributed across several computers in the network.

D. Fault-tolerance--Capability to provide a more graceful degradation of user service when a network failure occurs.

Projected Use	Very Beneficial	Beneficial	Somewhat Beneficial	Of Little Use	Of No Use
Peripheral Sharing	7	0	0	0	0
File Transfers	5	2	0	0	0
Software Tool Sharing	2	4	1	0	0
Access to CYBER 750	3	3	1	0	0
Access to AFITNET	3	4	0	0	0
Distributed Processing	2	1	2	2	0
Distributed Databases	2	0	2	3	0
Fault Tolerance	0	1	1	3	2

E. What group of things would you like to see implemented first?

Peripheral Sharing------7

File Transfers----------6

Software Tool Sharing---1

Access to CYBER 750-----1

Distributed Database----1

Section II Functional Requirements of DELNET

A. User-oriented Requirements

1. Throughput

a. For the courses that you will be teaching during the next four quarters, which computers do you expect to be utilized by your students?

b. What will be the average utilization?

c. What will be the maximum utilization?

Computer	Normal Usage	Maximum Usage
Nova/Eclipse	8 hours/day	24 hours/day
VAX-11/780	7 hours/day	10 hours/day
LSI-11s	10 hours/day	18 hours/day
Intel Series II	2 hours/day	3 hours/day
MIME	3 hours/day	4 hours/day

d. How would you see the network being utilized by your students?

File Transfers-------3

Peripheral Sharing---1

Access to CYBER 750--1

Simulations----------1

Interactive Mode-----1

2. Response Time

Do you forsee any projected uses of the network that require that the response time of the network to a set of commands not exceed some threshold?

No---------------------1

Yes, 60K file transfers from CYBER in less than 10 min

Yes, file transfers in 10-15 seconds

Yes, interactive command responses in 5-7 sec

Yes, interactive command responses in 2-3 sec

Yes, interactive command responses in less than 1 sec

Yes, inputs echoed in less than 0.5 sec

Yes, general response time should be less than 1 min

Yes, standard deviation of the response time distribution should less than 1/2 the mean response time

3. User Interface

a. Would symbolic device accessing be desirable? (Making the actual host that an I/O device is connected to transparent to the user)

Yes---7

No----0

b. Should the same be done for software tools?

Yes------------------5

No-------------------1

Should have option---1

c. What other features should the network present to the user?

Overall Transparency-------------------------4

Error Recovery-------------------------------1

User Helps-----------------------------------1

Option to Waive Transparency----------------1

System Disk Center---------------------------1

Database Access for Applications Programs---1

4. Security

a. Do you forsee the running of classified data or programs on the network, and if so at what classification?

Yes---0

No----7

b. Do you forsee the need to protect databases on the network from unauthorized access?

Yes---4

No----2

Maybe-1

Access to the network needs to be controlled--1

Varying levels of access are needed-----------1

5. Availability

a. What is the minimum percentage of time that you feel that the network should be available?

b. What is the reason for giving the above availability?

99.9%----1	(This should be attainable)
90%------3	(Research needs for digital signal processing, should approach availability of machines)
80%------1	(Gives some slack)
60%------1	(Sufficient to meet requirements of digital laboratory)

6. Other User-oriented Requirements

Self-answering modem connected to one node----1

B. Design-oriented Requirements

These requirements are not as user-oriented and you need only comment on those that you have interest in.

1. Flexibility

a. What changes do you see being made to the network during the next few years?

More hosts and devices being added to it	6
An on-line 5 Mbit storage device being added	1
Being used more for non-DEL purposes	1
More networks being accessible from DELNET	1
Workload changing from mainly file transfers to more interactive command use	1
More software development systems being added	1

b. How important do you feel that it is for DELNET to be easily reconfigurable with respect to the following:

Network Subcomponents	Very	Somewhat	Not	Pedagogical Only
New Computers and Devices	7	0	0	0
New Topologies	0	1	2	4
New Protocols	1	1	1	4
New Transmission Medium	2	1	1	3

Important to be able to vary between serial and parallel-1

2. Performance Monitoring

What performance monitoring capabilities should DELNET have?

Collect accounting data	3
Collect node statistics	3
Hardware monitors	2
Software monitors	2
As much as possible	2
Monitor network status	1
Detect network bottlenecks	1
Performance monitoring node	1

3. Pedagogical

One of the purposes of the network is to provide a learning and experimental tool for the students in such areas as digital hardware design, performance evaluation, applications programming, computer networking, and operating systems design.

What specific requirements do you feel are needed to insure that these goals are met? (For instance, one such suggested requirement is that DELNET provide the flexibility for implementation of different communications protocols, information formats, and network or subnetwork topologies).

Flexibility	4
Performance Monitoring	3
Testing a fiber optic link	1

4. Distributed Processing Language

What language(s) would you like to see implemented on all or most of the hosts if a distributed processing capability is implemented?

Pascal-------7 (one specified UCSD Pascal)

Ada----------6

FORTRAN------5

BASIC--------1

Ada has powerful control structures for distributed processing-1

5. Other design-oriented requirements

Are there any other design oriented requirements that should be addressed?

None----7

C. How would you rank the functional requirements for the above areas?

Area	Not Applicable	Marginal.y Applicable	Applicable	Very Applicable	Essential
Throughput	1	1	3	2	0
Response Time	0	1	2	3	1
User Interface	0	0	1	3	3
Security	2	1	1	1	2
Availability	0	1	1	3	2
Flexibility	0	0	0	4	3
Performance Monitoring	0	1	0	4	2
Pedagogical	0	1	0	4	2
Distributed Processing Language	0	2	2	3	0

Section III Other Comments

What other comments or suggestions do you have concerning the projected uses and/or functional requirements of DELNET?

1. It would be nice to have a general purpose node for checking out student projects with network resources.

2. The network software must be interrupt-driven.

3. It would be nice to be able to interrogate the network to find a file that was sent.

4. The network should not be used for general software development.

5. The requirement to use all nodes is probably only applicable one percent of the time.

6. It should be easy to initialize the network with an arbitrary subset of the nodes available on the network.

7. The number of terminals and their locations should be addressed.

8. Top-down decomposition and structured analysis should be used in the design.

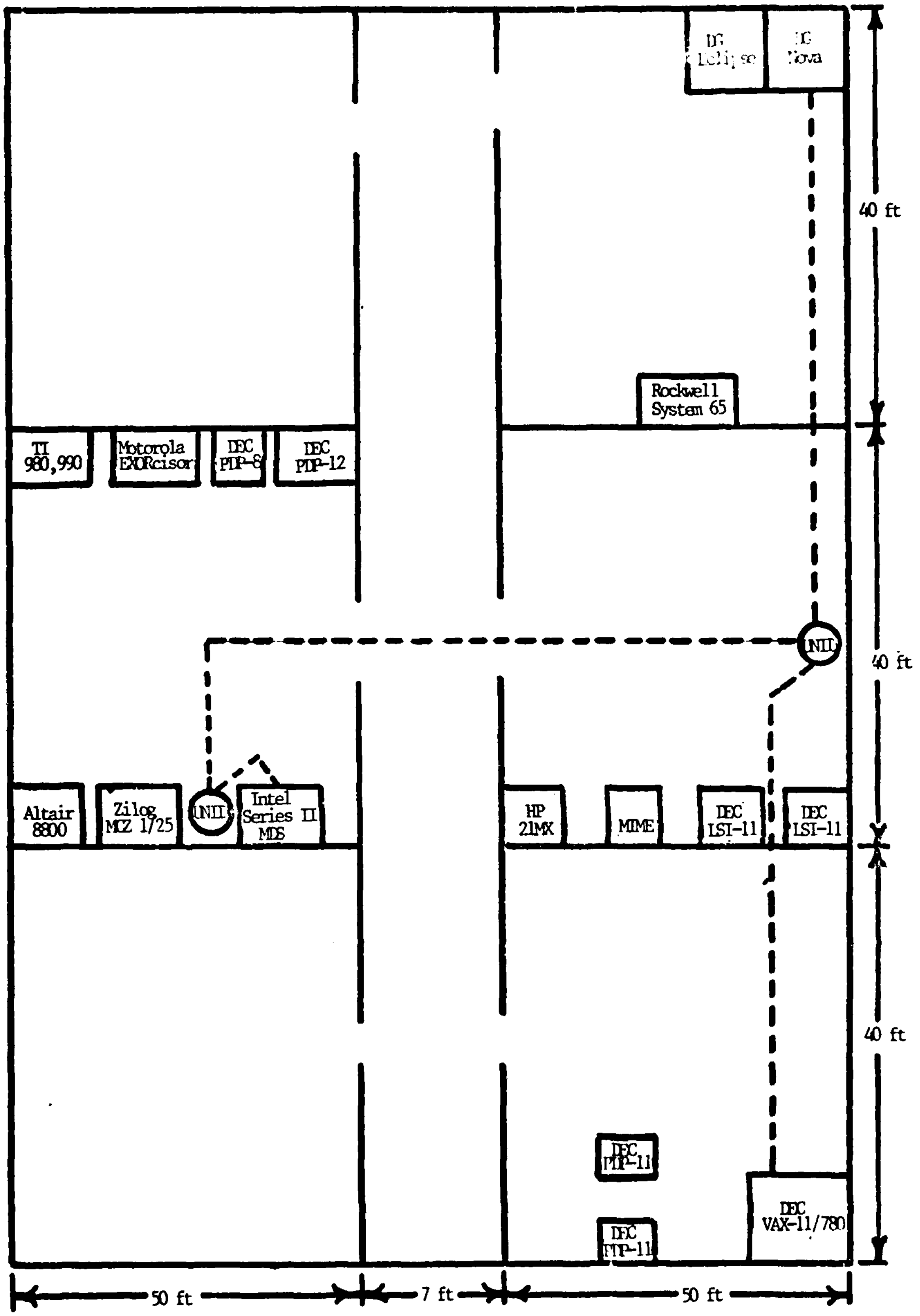
List of Users Who Were Interviewed

User's Name	Position on Faculty	Area of Interest
Capt Roie Black	Math Assistant Prof	Compiler Theory, Numerical Analysis
Dr. Thomas Hartrum	EE Assistant Prof	Databases, Performance Monitoring, Operating Systems
Capt Larry Kizer	EE Instructor	Digital Signal Processing
Dr. Gary Lamont	EE Professor	Computer Engineering/ Computer Science
Capt James Moore	EE Instructor	Computer Networking
Maj Alan Ross	EE Assistant Prof	Computer Architecture
Maj James Rutledge	EE Assistant Prof	Software Engineering
Capt Walter Seward	EE Assistant Prof	Computer Architecture, Computer Networking
Maj Michael Wirth	Math Instructor	AFITNET design

Appendix B

Digital Engineering Laboratory Floor Layout Diagram

This appendix contains a floor layout diagram of the Digital Engineering Laboratory at the Air Force Institute of Technology School of Engineering. The locations of all computers with semi-permanent locations are shown as well as the proposed locations of the Universal Network Interface Devices (UNIDs). Finally, the computer links that will be initially installed to the UNIDs are also shown.



DG Eclipse
DG Nova
40 ft
Rockwell System 65
TI 980,990
Motorola EXORciser
DEC PDP-8
DEC PDP-12
NIU
40 ft
Altair 8800
Zilog MCZ 1/25
NIU
Intel Series II MDS
HP 21MX
MIME
DEC LSI-11
DEC LSI-11
40 ft
DEC PDP-11
DEC PDP-11
DEC VAX-11/780
50 ft
7 ft
50 ft

Appendix C

Structured Specification

This appendix contains the structured specification of the software requirements for DELNET. First, the complete set of data flow diagrams (Figures 1-32 in the main body) is included. These are followed by the data dictionaries for the high-level protocol, the host-to-host protocol, the network protocol, and the link protocol.

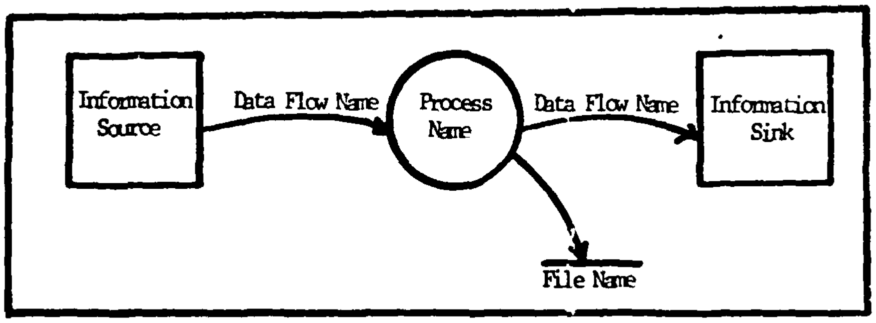


Figure 1 DFD Components

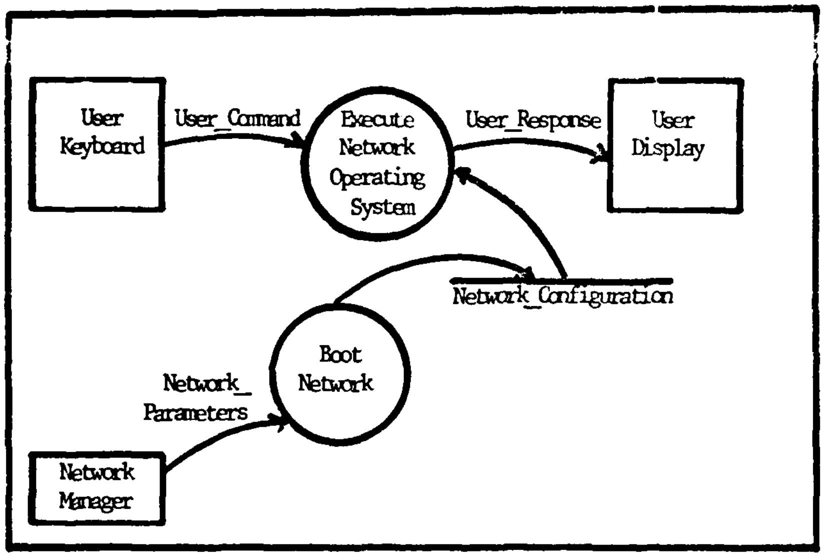


Figure 2 Context Diagram

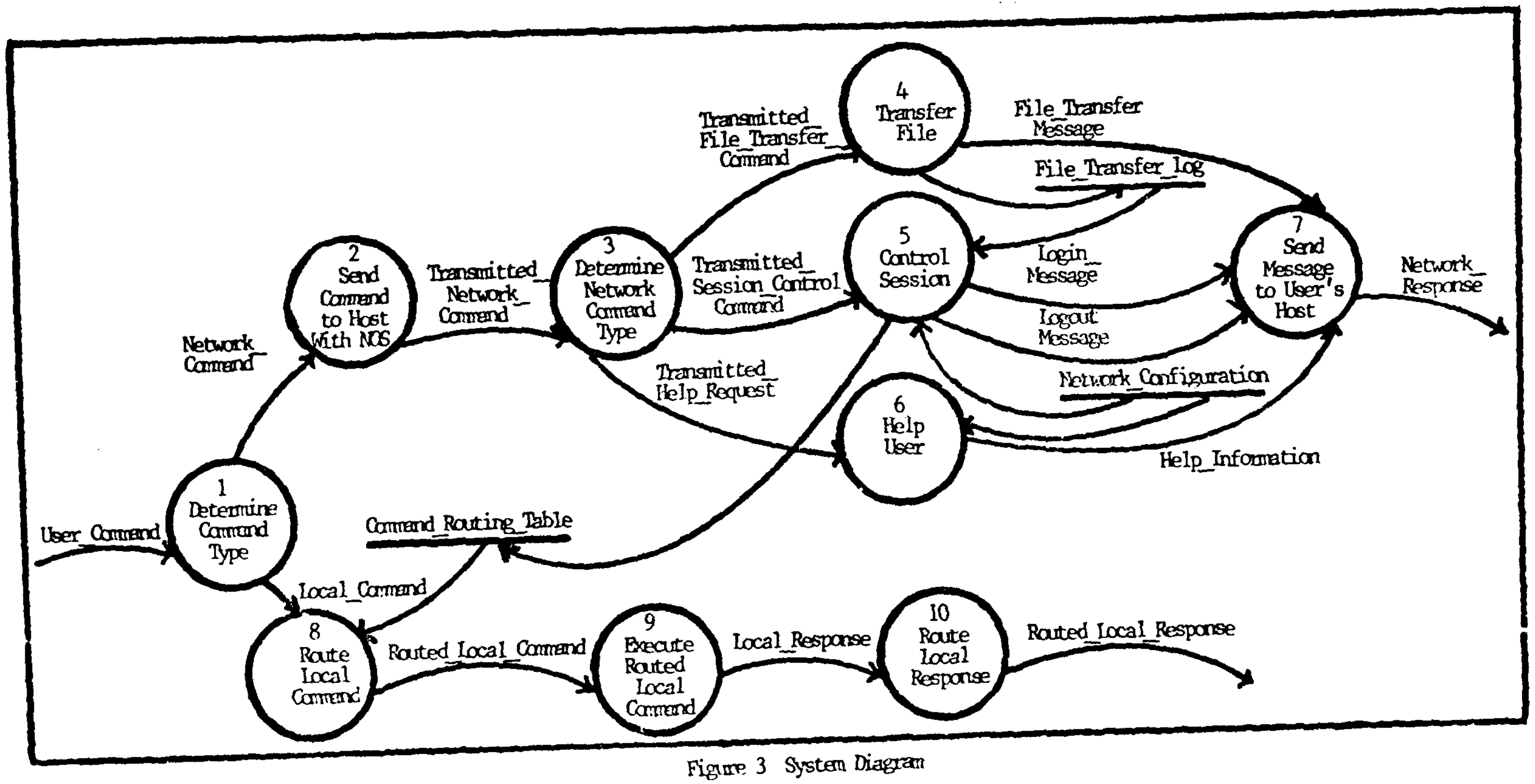


Figure 3 System Diagram

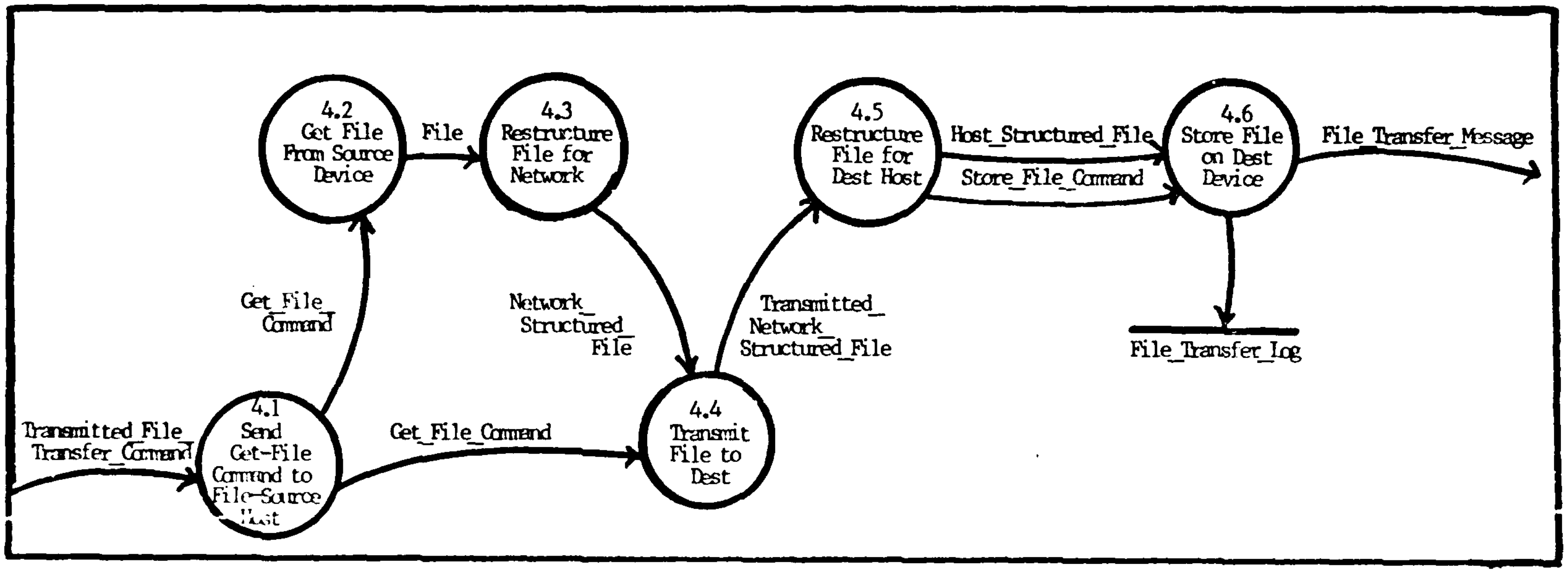

4.1 Send Get-File Command to File-Source Host
4.2 Get File From Source Device
4.3 Restructure File for Network
4.4 Transmit File to Dest
4.5 Restructure File for Dest Host
4.6 Store File on Dest Device
Transmitted File Transfer_Command
Get_File_Command
Get_File_Command
File
Network_Structured_File
Transmitted_Network_Structured_File
Host_Structured_File
Store_File_Command
File_Transfer_Message
File_Transfer_Log

Figure 4 Transfer File (4.0) DFD

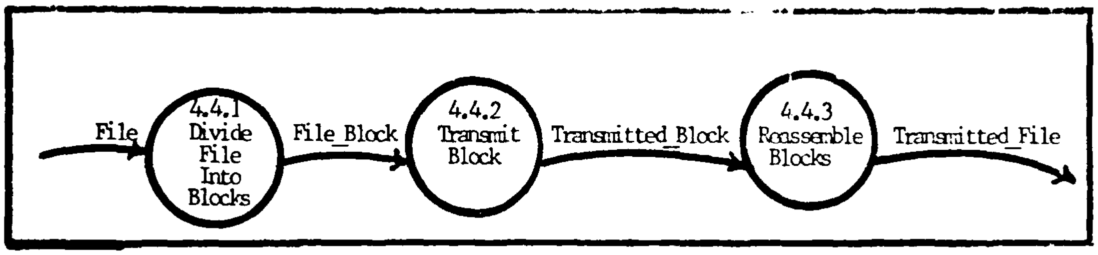


Figure 5 Transmit File (4.4) DFD

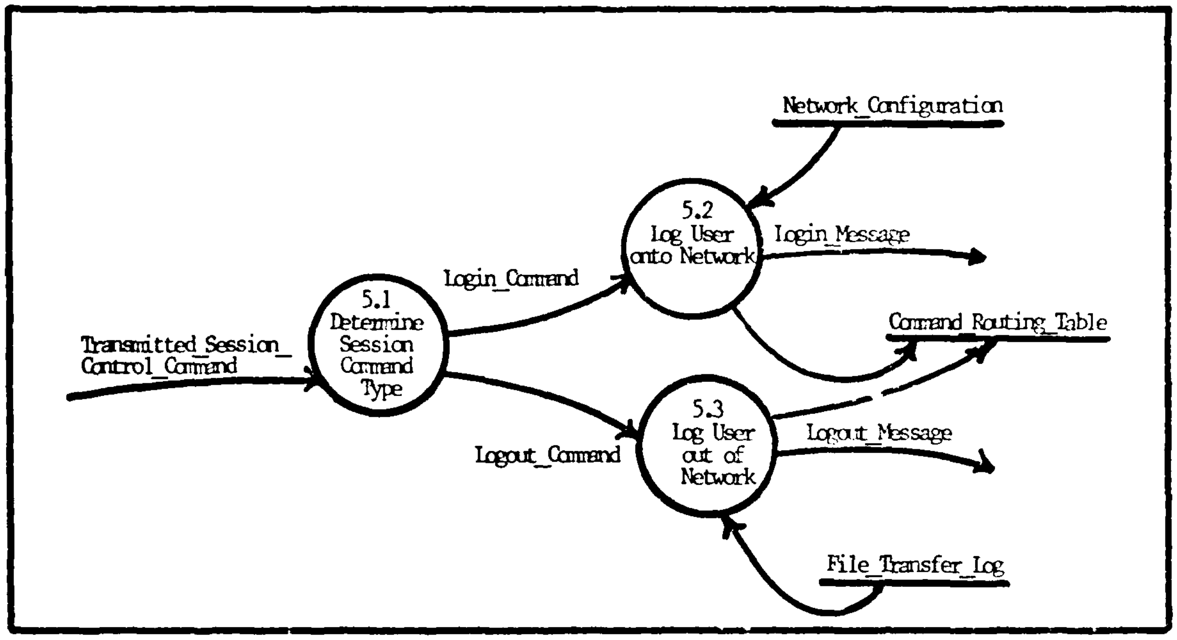
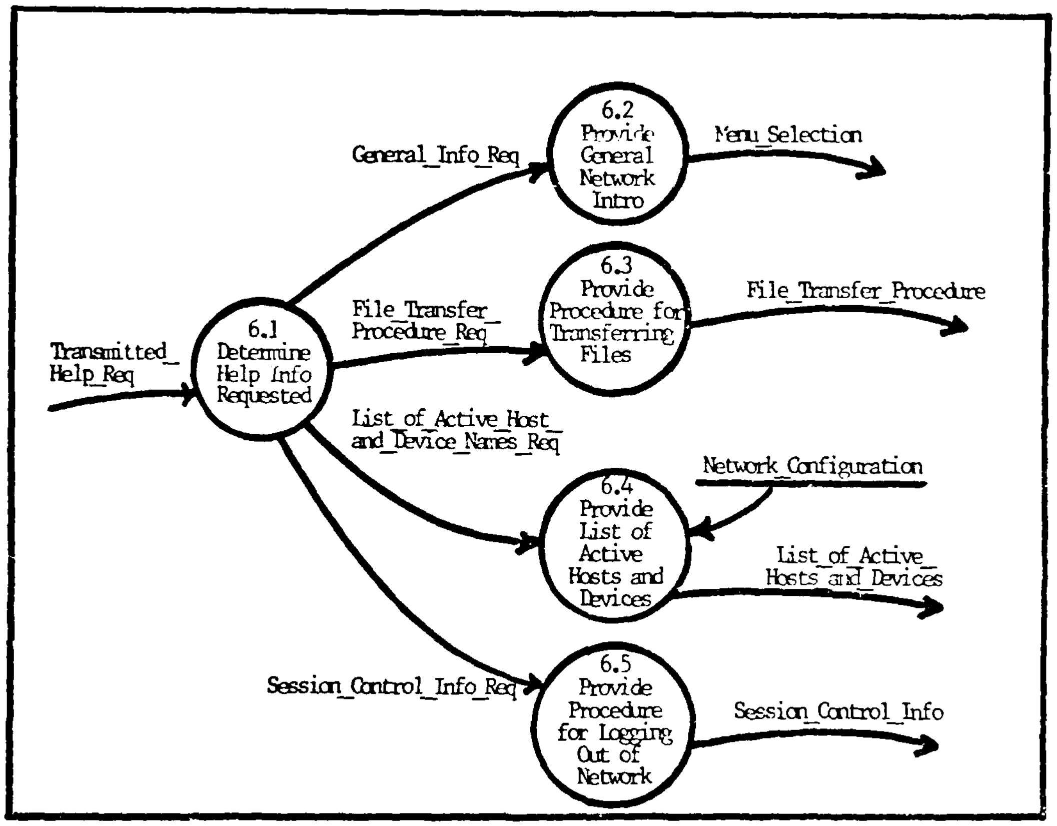

Network_Configuration
5.2 Log User onto Network
Login_Message
Login_Command
5.1 Determine Session Command Type
Command_Routing_Table
Transmitted_Session_Control_Command
5.3 Log User out of Network
Logout_Message
Logout_Command
File_Transfer_Log

Figure 6 Control Session (5.0) DFD



6.1 Determine Help Info Requested
6.2 Provide General Network Intro
6.3 Provide Procedure for Transferring Files
6.4 Provide List of Active Hosts and Devices
6.5 Provide Procedure for Logging Out of Network
Transmitted_Help_Req
General_Info_Req
File_Transfer_Procedure_Req
List_of_Active_Host_and_Device_Names_Req
Session_Control_Info_Req
Menu_Selection
File_Transfer_Procedure
Network_Configuration
List_of_Active_Hosts_and_Devices
Session_Control_Info

Figure 7 Help User (6.0) DFD

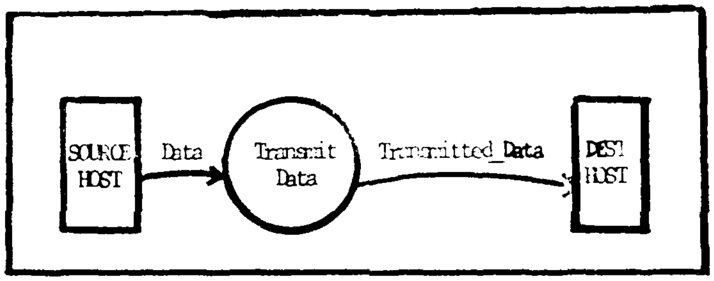


Figure 8 Host-to-Host Context Diagram

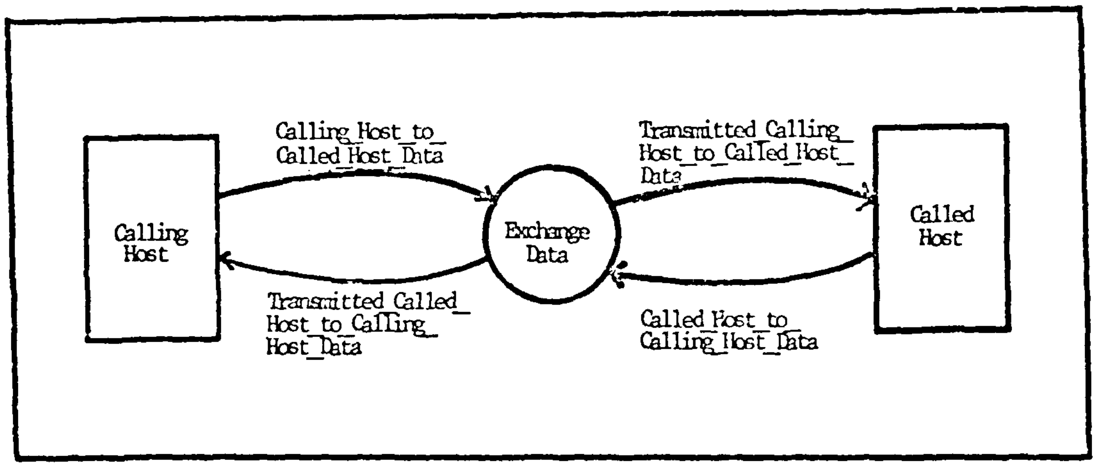

Calling Host
Calling_Host_to_Called_Host_Data
Exchange Data
Transmitted_Calling_Host_to_Called_Host_Data
Called Host
Transmitted_Called_Host_to_Calling_Host_Data
Called_Host_to_Calling_Host_Data

Figure 9 X.25 Level 3 Context Diagram

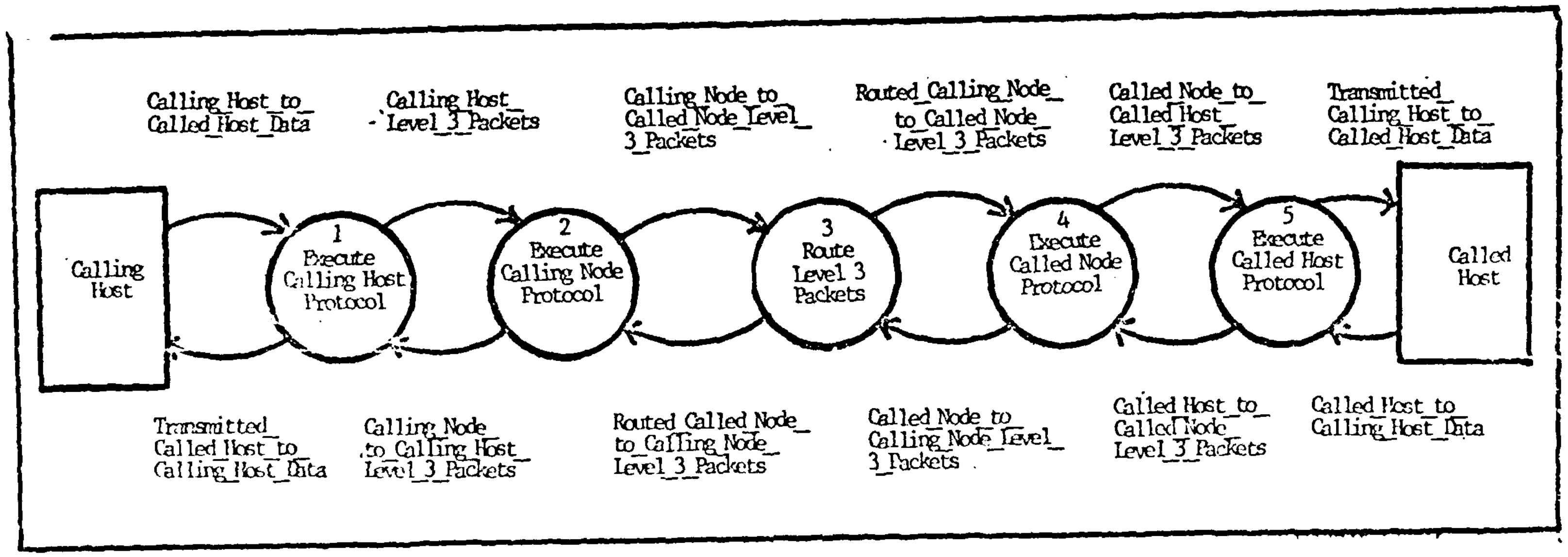

Calling Host_to_ Called_Host_Data
Calling Host_ Level_3_Packets
Calling_Node_to Called_Node_Level_ 3_Packets
Routed_Calling_Node_ to_Called_Node_ Level_3_Packets
Called_Node_to_ Called_Host_ Level_3_Packets
Transmitted_ Calling_Host_to_ Called_Host_Data
Calling Host
1 Execute Calling Host Protocol
2 Execute Calling Node Protocol
3 Route Level 3 Packets
4 Execute Called Node Protocol
5 Execute Called Host Protocol
Called Host
Transmitted_ Called_Host_to_ Calling_Host_Data
Calling_Node_ to_Calling_Host_ Level_3_Packets
Routed_Called_Node_ to_Calling_Node_ Level_3_Packets
Called_Node_to Calling_Node_Level_ 3_Packets
Called_Host_to_ Called_Node_ Level_3_Packets
Called_Host_to Calling_Host_Data

Figure 10 X.25 Level 3 Overview DFD

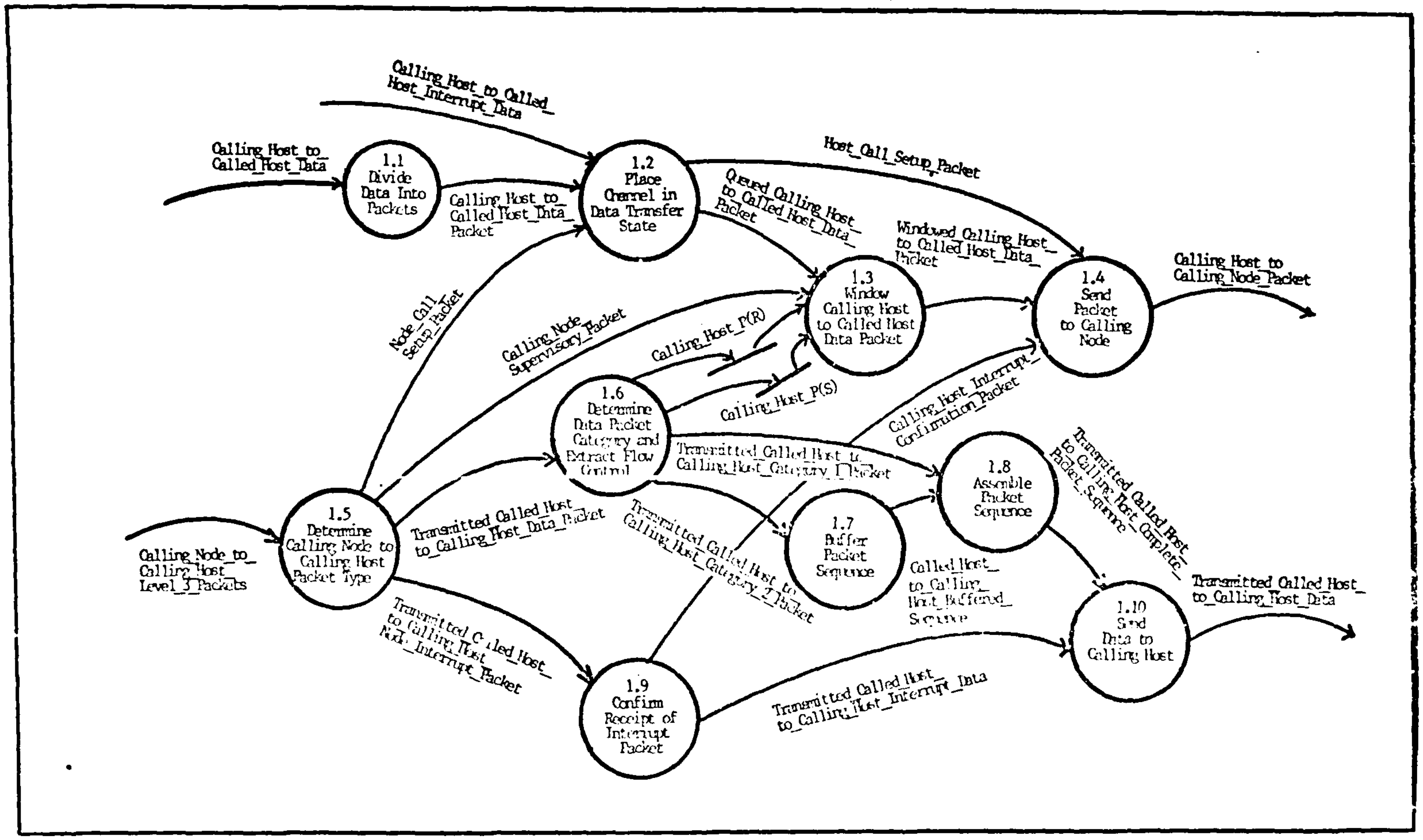

Calling_Host_to_Called_Host_Data
1.1 Divide Data Into Packets
Calling_Host_to_Called_Host_Interrupt_Data
Calling_Host_to_Called_Host_Data_Packet
1.2 Place Channel in Data Transfer State
Host_Call_Setup_Packet
Queued_Calling_Host_to_Called_Host_Data_Packet
Windowed_Calling_Host_to_Called_Host_Data_Packet
1.3 Window Calling Host to Called Host Data Packet
1.4 Send Packet to Calling Node
Calling_Host_to Calling_Node_Packet
Node_Call_Setup_Packet
Calling_Node_Supervisory_Packet
Calling_Host_P(R)
Calling_Host_P(S)
1.6 Determine Data Packet Category and Extract Flow Control
Calling_Host_Interrupt_Confirmation_Packet
Calling_Node_to_Calling_Host_Level_3_Packets
1.5 Determine Calling Node to Calling Host Packet Type
Transmitted_Called_Host_to_Calling_Host_Data_Packet
Transmitted_Called_Host_to_Calling_Host_Category_1_Packet
Transmitted_Called_Host_to_Calling_Host_Category_2_Packet
1.7 Buffer Packet Sequence
1.8 Assemble Packet Sequence
Transmitted_Called_Host_to_Calling_Host_Complete_Packet_Sequence
Called_Host_to_Calling_Host_Buffered_Sequence
Transmitted_Called_Host_to_Calling_Host_Data
1.10 Send Data to Calling Host
Transmitted_Called_Host_to_Calling_Host_Interrupt_Packet
1.9 Confirm Receipt of Interrupt Packet
Transmitted_Called_Host_to_Calling_Host_Interrupt_Data

Figure 11 Execute Calling Host Protocol (1) DFD

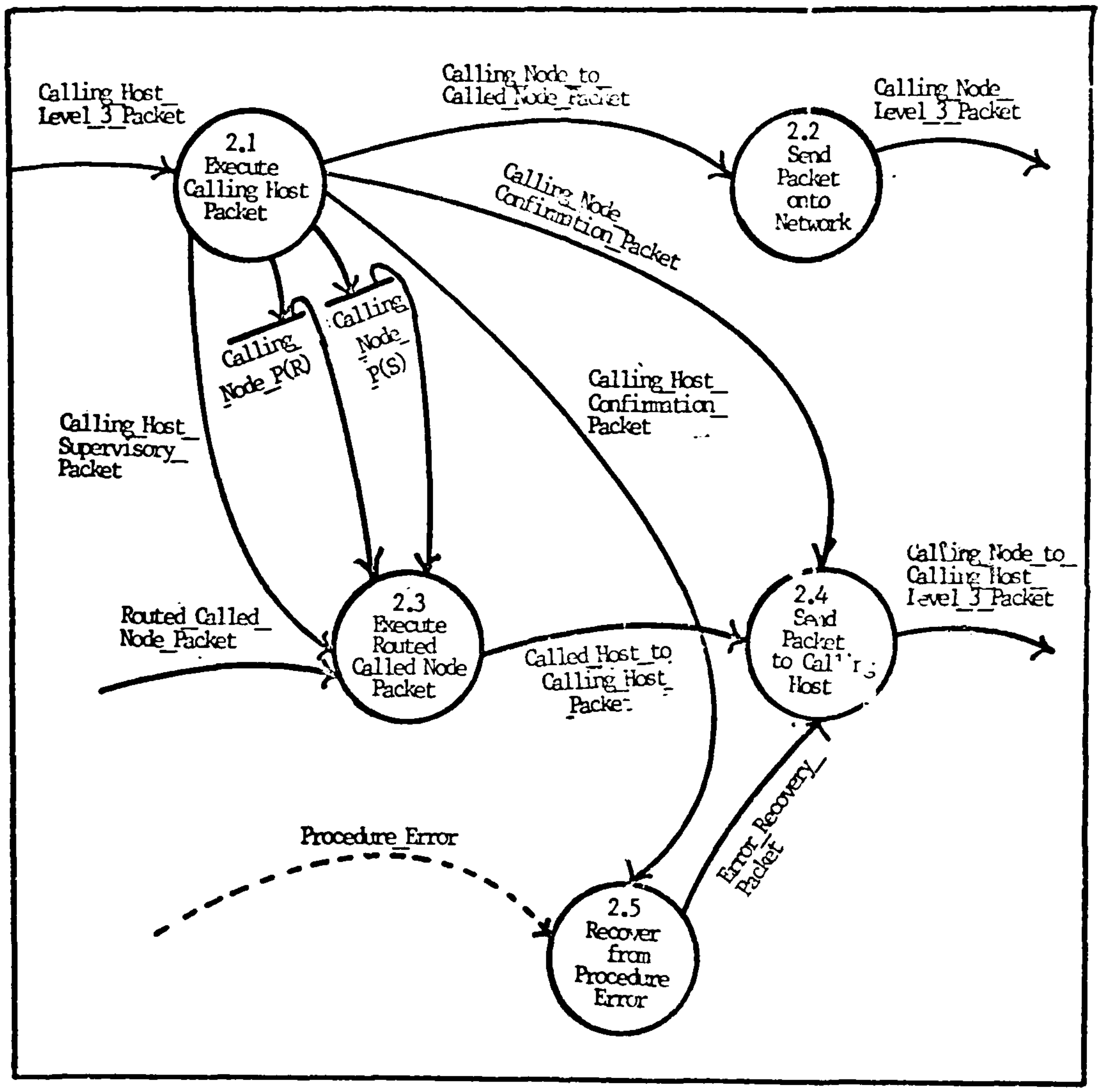

Calling_Host_ Level_3_Packet
2.1 Execute Calling Host Packet
Calling_Node_to_ Called_Node_Packet
2.2 Send Packet onto Network
Calling_Node_ Level_3_Packet
Calling_Node_ Confirmation_Packet
Calling_ Node_P(R)
Calling_ Node_ P(S)
Calling_Host_ Confirmation_ Packet
Calling_Host_ Supervisory_ Packet
Routed_Called_ Node_Packet
2.3 Execute Routed Called Node Packet
Called_Host_to_ Calling_Host_ Packet
2.4 Send Packet to Calling Host
Calling_Node_to_ Calling_Host_ Level_3_Packet
Procedure_Error
2.5 Recover from Procedure Error
Error_Recovery_ Packet

Figure 12 Execute Calling Node Protocol (2) DFD

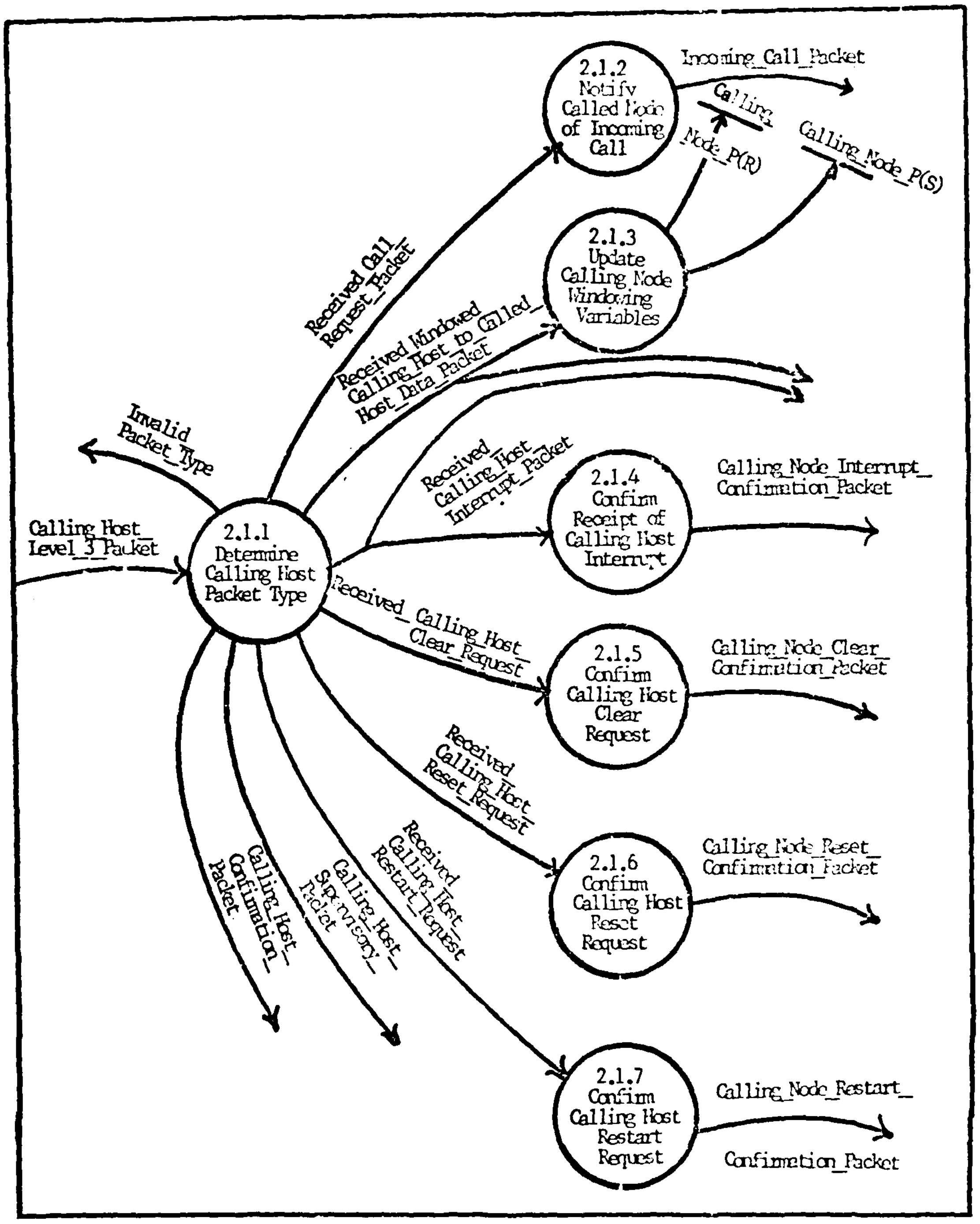

2.1.1 Determine Calling Host Packet Type
2.1.2 Notify Called Node of Incoming Call
2.1.3 Update Calling Node Windowing Variables
2.1.4 Confirm Receipt of Calling Host Interrupt
2.1.5 Confirm Calling Host Clear Request
2.1.6 Confirm Calling Host Reset Request
2.1.7 Confirm Calling Host Restart Request
Calling_Host_Level_3_Packet
Invalid Packet_Type
Received_Call_Request_Packet
Received_Windowed_Calling_Host_to_Called_Host_Data_Packet
Received_Calling_Host_Interrupt_Packet
Received_Calling_Host_Clear_Request
Received_Calling_Host_Reset_Request
Received_Calling_Host_Restart_Request
Calling_Host_Supervisory_Packet
Calling_Host_Confirmation_Packet
Incoming_Call_Packet
Calling_Node_P(R)
Calling_Node_P(S)
Calling_Node_Interrupt_Confirmation_Packet
Calling_Node_Clear_Confirmation_Packet
Calling_Node_Reset_Confirmation_Packet
Calling_Node_Restart_Confirmation_Packet

Figure 13 Execute Calling Host Packet (2.1) DFD

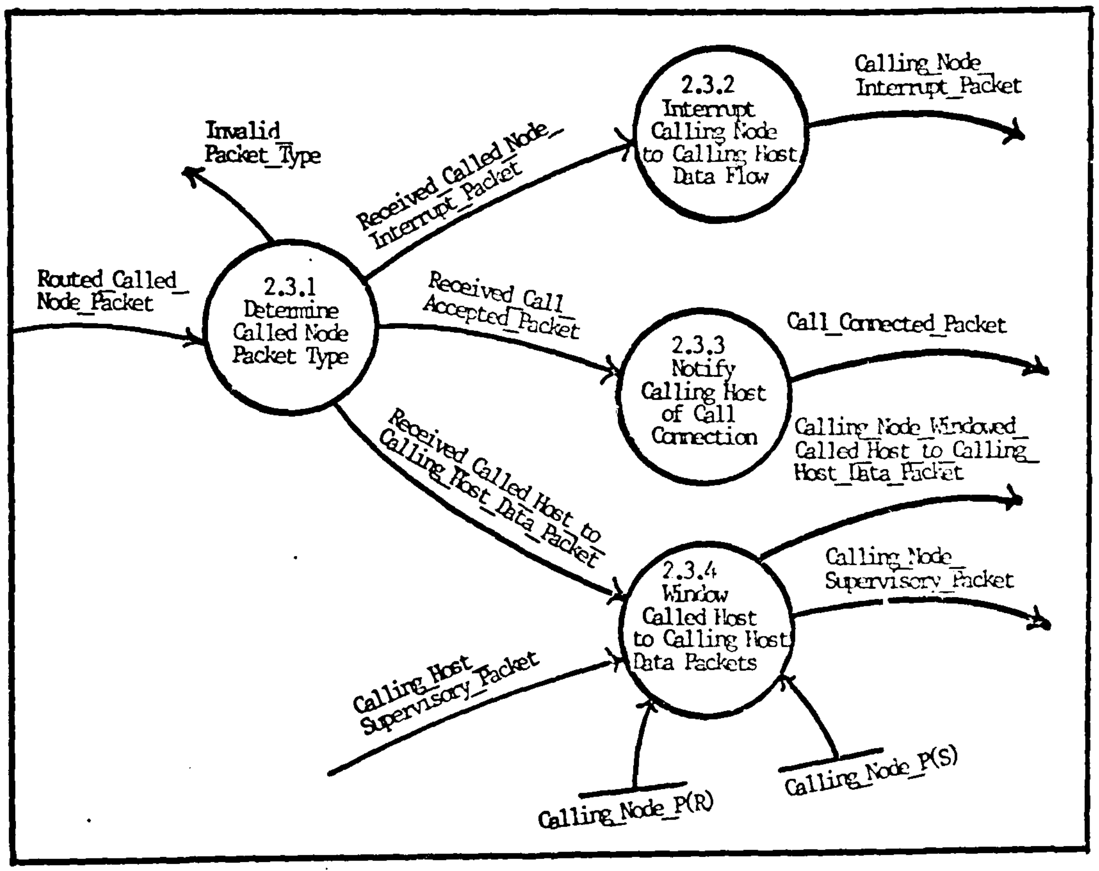

Invalid Packet_Type
Routed_Called_ Node_Packet
2.3.1 Determine Called Node Packet Type
Received_Called_Node_ Interrupt_Packet
2.3.2 Interrupt Calling Node to Calling Host Data Flow
Calling_Node_ Interrupt_Packet
Received_Call_ Accepted_Packet
2.3.3 Notify Calling Host of Call Connection
Call_Connected_Packet
Calling_Node_Windowed_ Called_Host_to_Calling_ Host_Data_Packet
Received_Called_Host_to_ Calling_Host_Data_Packet
2.3.4 Window Called Host to Calling Host Data Packets
Calling_Node_ Supervisory_Packet
Calling_Host_ Supervisory_Packet
Calling_Node_P(R)
Calling_Node_P(S)

Figure 14 Execute Routed Called Node Packet (2.3) DFD

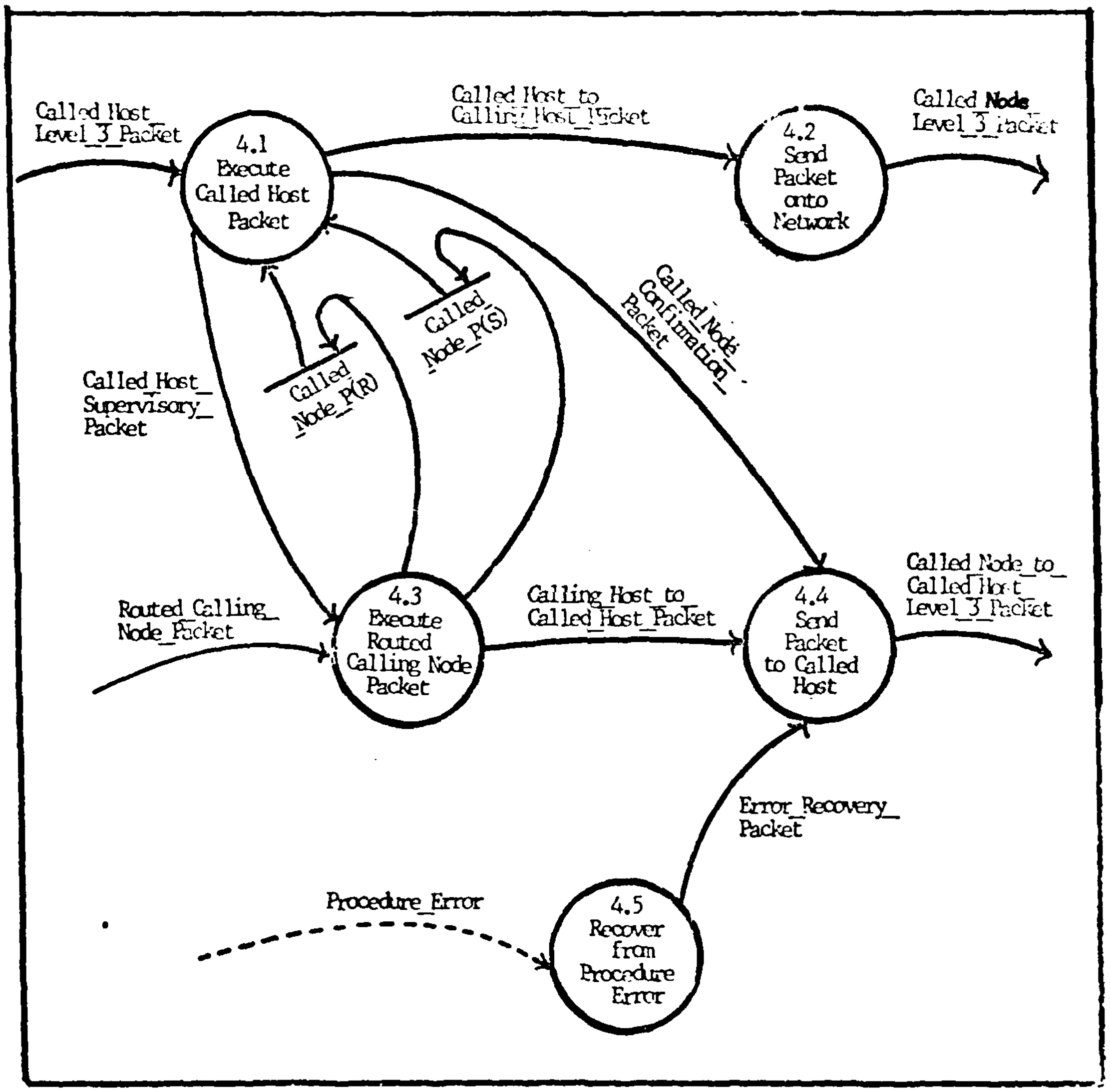
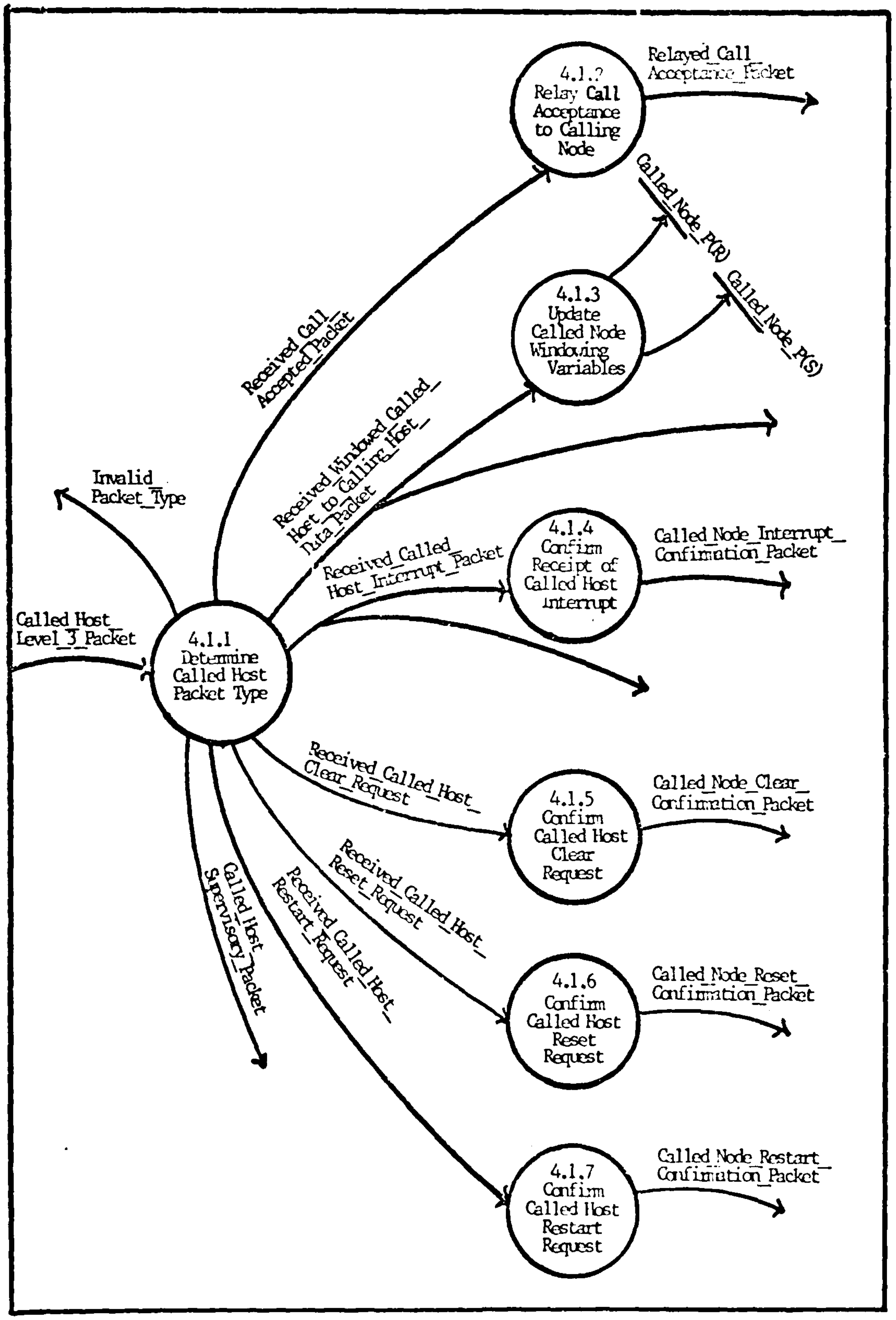


Figure 15 Execute Called Node Protocol (4) DFD



4.1.2 Relay Call Acceptance to Calling Node
Relayed_Call_Acceptance_Packet
Called_Node_P(R)
4.1.3 Update Called Node Windowing Variables
Called_Node_P(S)
Received_Call_Accepted_Packet
Received_Windowed_Called_Host_to_Calling_Host_Data_Packet
Invalid_Packet_Type
4.1.4 Confirm Receipt of Called Host Interrupt
Received_Called_Host_Interrupt_Packet
Called_Node_Interrupt_Confirmation_Packet
Called_Host_Level_3_Packet
4.1.1 Determine Called Host Packet Type
Received_Called_Host_Clear_Request
4.1.5 Confirm Called Host Clear Request
Called_Node_Clear_Confirmation_Packet
Received_Called_Host_Reset_Request
4.1.6 Confirm Called Host Reset Request
Called_Node_Reset_Confirmation_Packet
Called_Host_Supervisory_Packet
Received_Called_Host_Restart_Request
4.1.7 Confirm Called Host Restart Request
Called_Node_Restart_Confirmation_Packet

Figure 16 Execute Called Host Packet (4.1) DFD

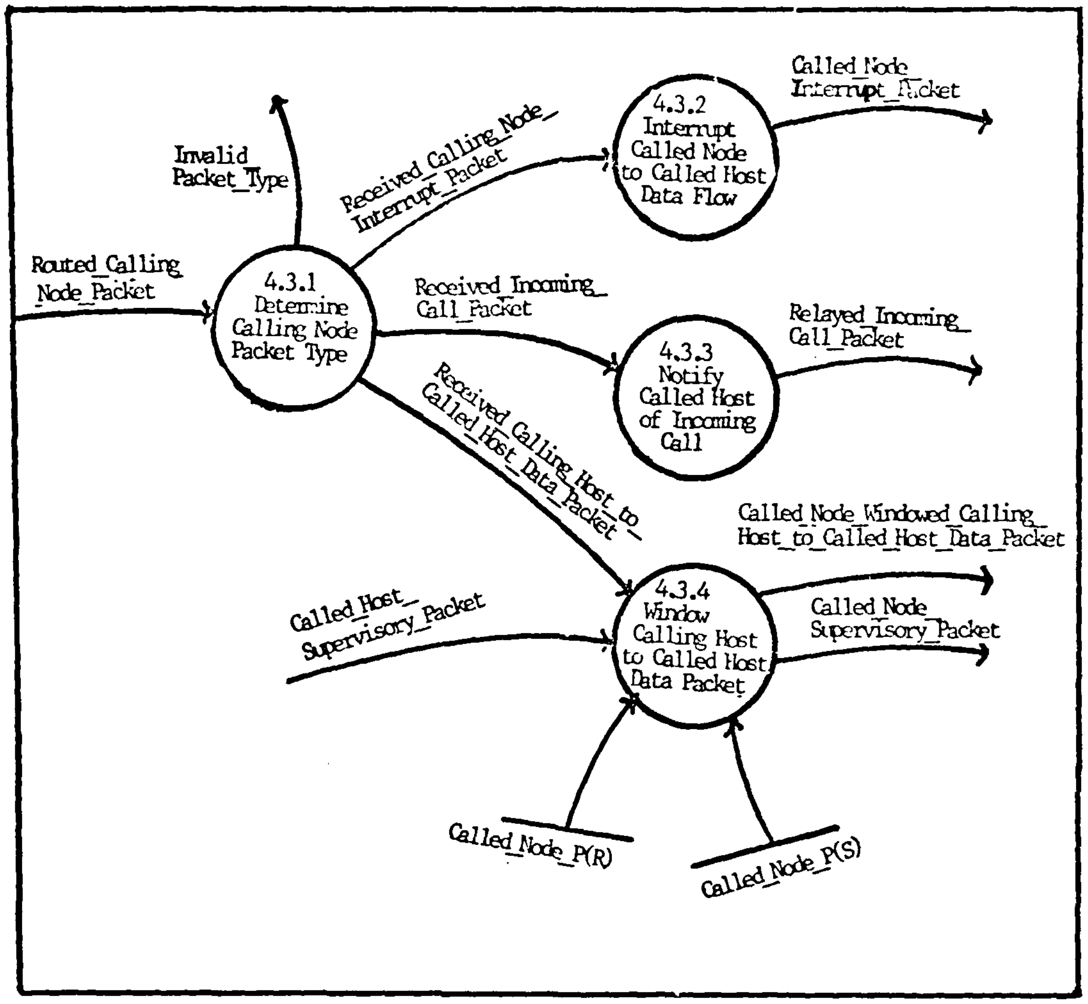


Figure 17 Execute Routed Calling Node Packet (4.3) DFD

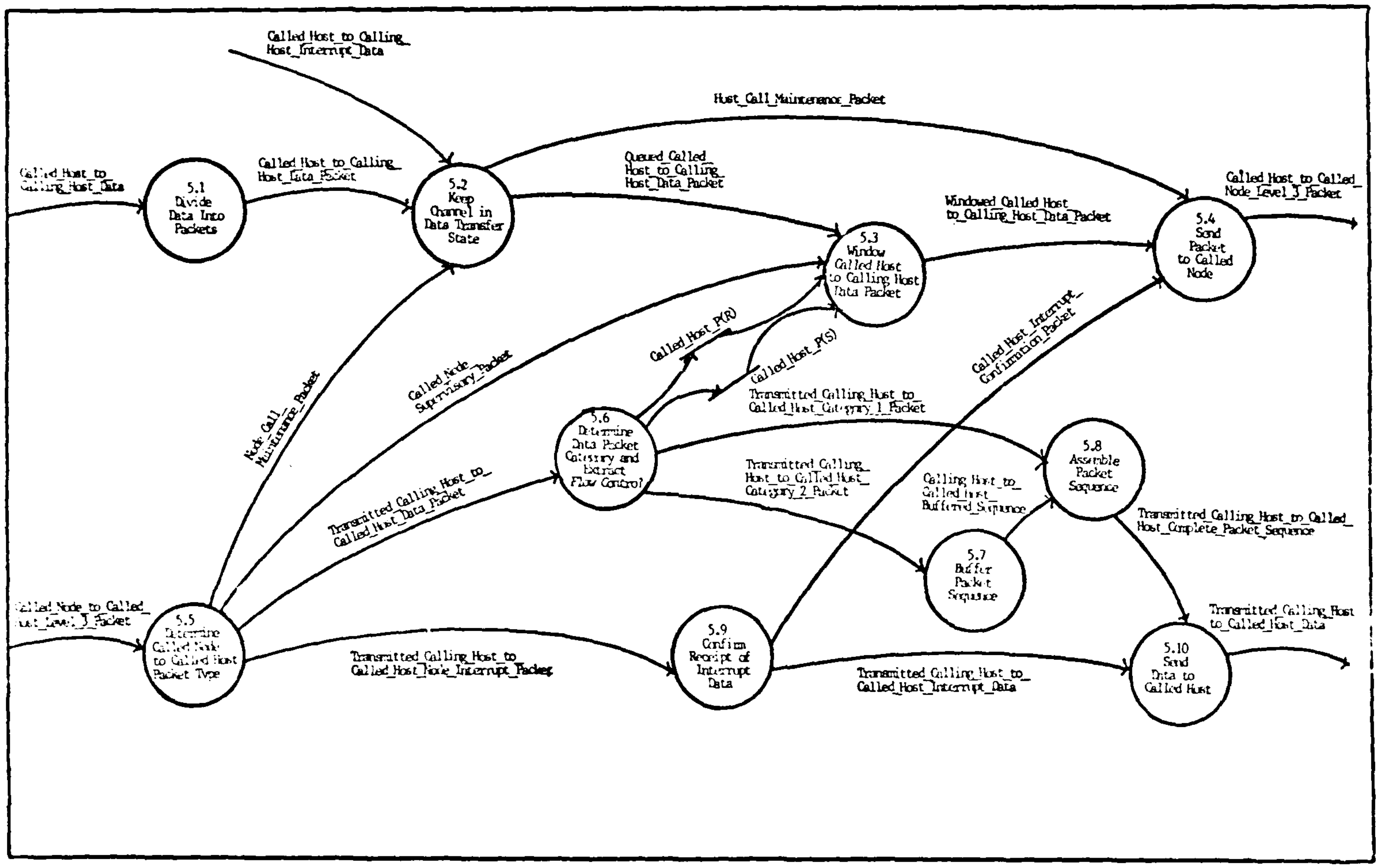

Called_Host_to_Calling_Host_Interrupt_Data
Host_Call_Maintenance_Packet
Called_Host_to_Calling_Host_Data
5.1 Divide Data Into Packets
Called_Host_to_Calling_Host_Data_Packet
5.2 Keep Channel in Data Transfer State
Queued_Called_Host_to_Calling_Host_Data_Packet
5.3 Window Called Host to Calling Host Data Packet
Windowed_Called_Host_to_Calling_Host_Data_Packet
5.4 Send Packet to Called Node
Called_Host_to_Called_Node_Level_3_Packet
Node_Call_Maintenance_Packet
Called_Node_Supervisory_Packet
Called_Host_P(R)
Called_Host_P(S)
Transmitted_Calling_Host_to_Called_Host_Category_1_Packet
Called_Host_Interrupt_Confirmation_Packet
5.6 Determine Data Packet Category and Extract Flow Control
Transmitted_Calling_Host_to_Called_Host_Data_Packet
Transmitted_Calling_Host_to_Called_Host_Category_2_Packet
Calling_Host_to_Called_Host_Buffered_Sequence
5.8 Assemble Packet Sequence
Transmitted_Calling_Host_to_Called_Host_Complete_Packet_Sequence
5.7 Buffer Packet Sequence
Called_Node_to_Called_Host_Level_3_Packet
5.5 Determine Called Node to Called Host Packet Type
Transmitted_Calling_Host_to_Called_Host_Node_Interrupt_Packet
5.9 Confirm Receipt of Interrupt Data
Transmitted_Calling_Host_to_Called_Host_Interrupt_Data
5.10 Send Data to Called Host
Transmitted_Calling_Host_to_Called_Host_Data

Figure 18 Execute Called Host Protocol (5) DFD

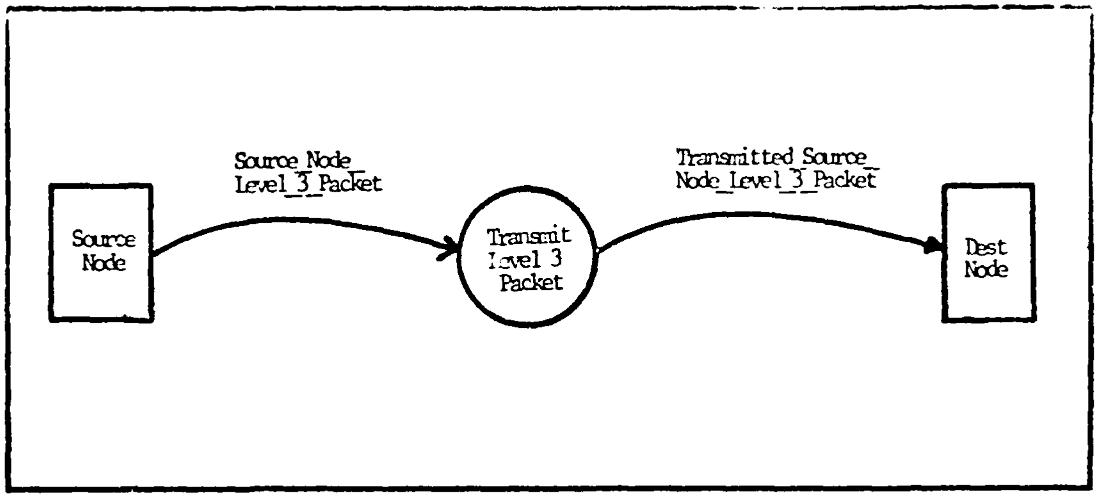

Source Node
Source_Node_Level_3_Packet
Transmit Level 3 Packet
Transmitted_Source_Node_Level_3_Packet
Dest Node

Figure 19. Network Protocol Context Diagram

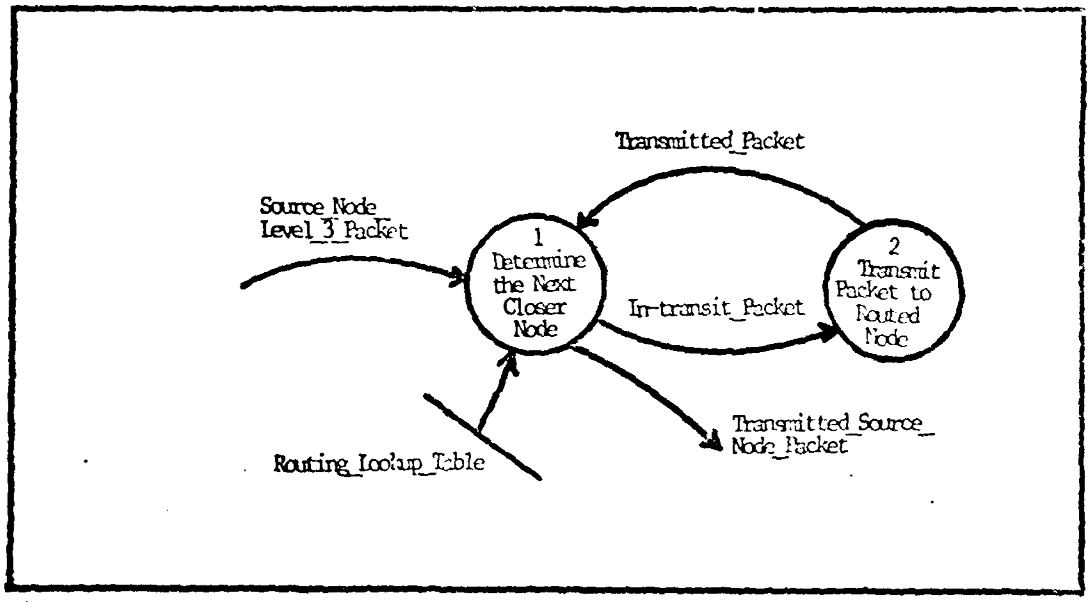


Figure 20 Network Protocol Overview DFD

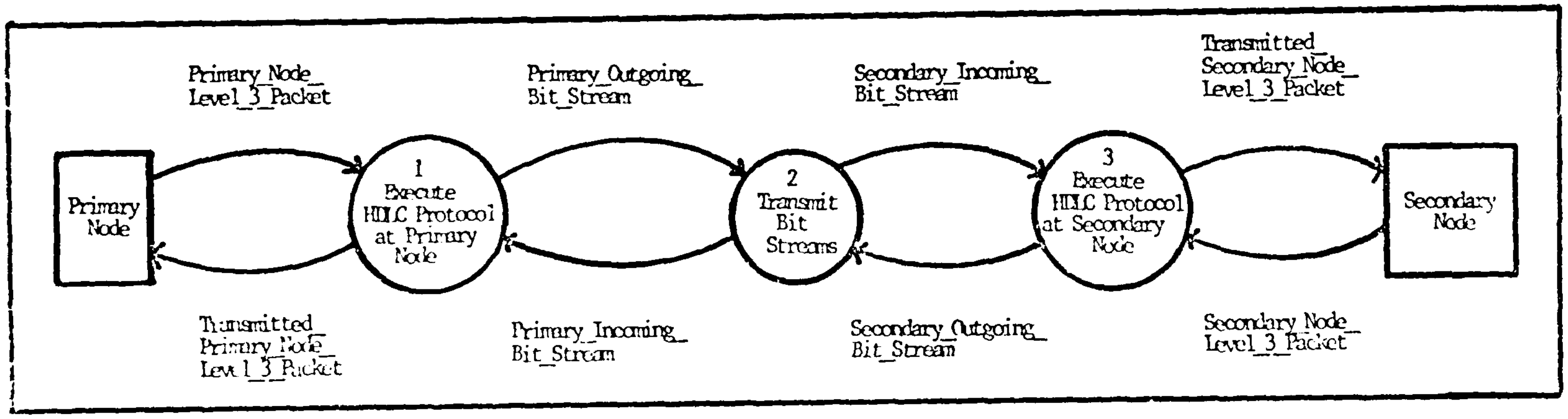

Primary Node
1 Execute HDLC Protocol at Primary Node
2 Transmit Bit Streams
3 Execute HDLC Protocol at Secondary Node
Secondary Node
Primary_Node_Level_3_Packet
Transmitted_Primary_Node_Level_3_Packet
Primary_Outgoing_Bit_Stream
Primary_Incoming_Bit_Stream
Secondary_Incoming_Bit_Stream
Secondary_Outgoing_Bit_Stream
Transmitted_Secondary_Node_Level_3_Packet
Secondary_Node_Level_3_Packet

Figure 21 X.25 Level 2 (HDLC) Overview DFD

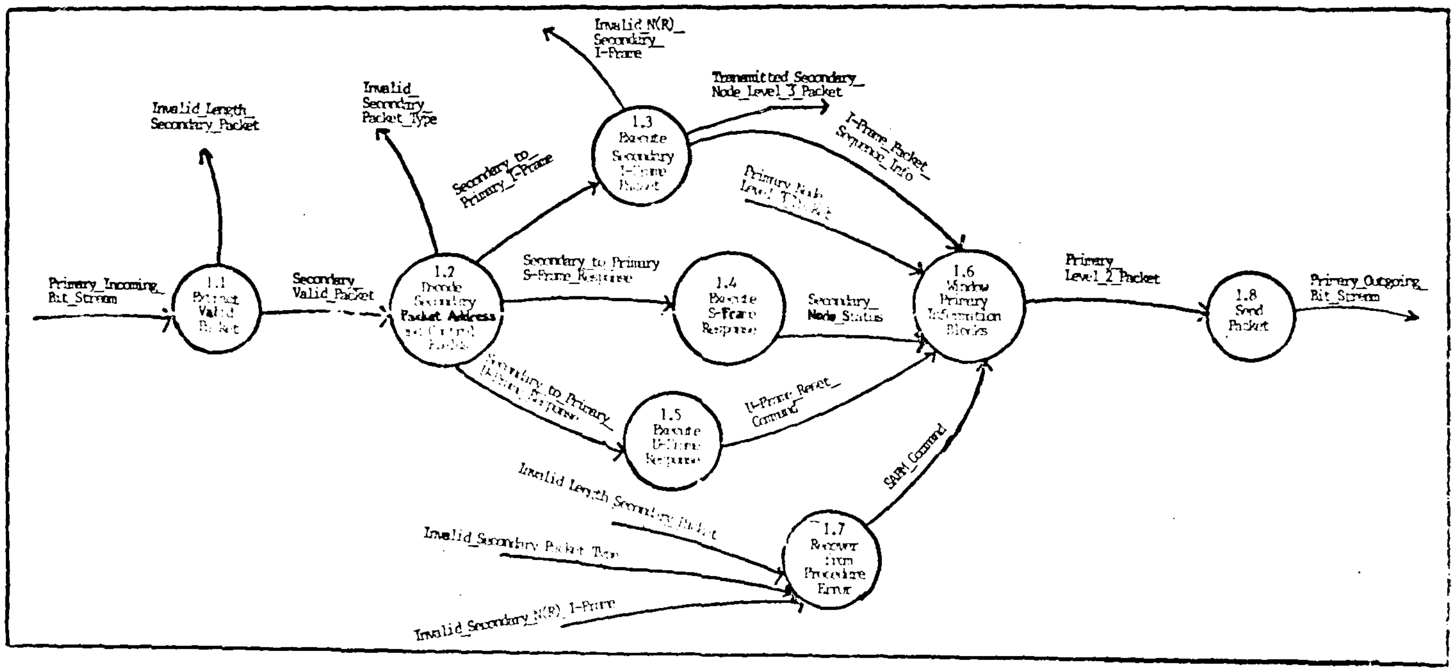

Primary_Incoming_ Bit_Stream
1.1 Extract Valid Packet
Invalid_Length_ Secondary_Packet
Secondary_ Valid_Packet
1.2 Decode Secondary Packet Address and Control Fields
Invalid_ Secondary_ Packet_Type
Secondary_to_ Primary_I-Frame
Secondary_to_Primary S-Frame_Response
Secondary_to_Primary_ U-Frame_Response
1.3 Execute Secondary I-Frame Packet
Invalid_N(R)_ Secondary_ I-Frame
Transmitted_Secondary_ Node_Level_3_Packet
I-Frame_Packet_ Sequence_Info
Primary_Node_ Level_3_Packet
1.4 Execute S-Frame Response
Secondary_ Node_Status
1.5 Execute U-Frame Response
U-Frame_Reset_ Command
1.6 Window Primary Information Blocks
SARM_Command
1.7 Recover from Procedure Error
Invalid Length Secondary Packet
Invalid_Secondary_Packet_Type
Invalid_Secondary_N(R)_I-Frame
Primary_ Level_2_Packet
1.8 Send Packet
Primary_Outgoing_ Bit_Stream

Figure 22 Execute HDLC Protocol at Primary Node (1) DFD

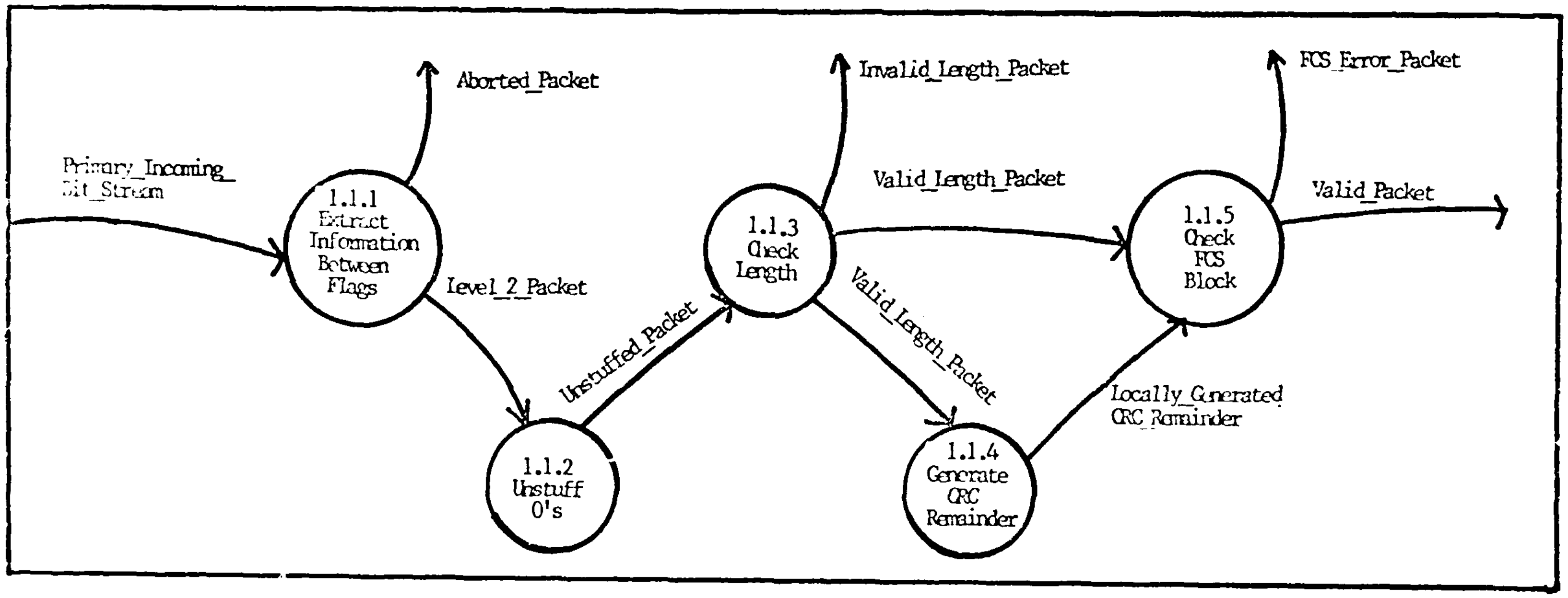


Figure 23 Extract Valid Packet (1.1) DFD

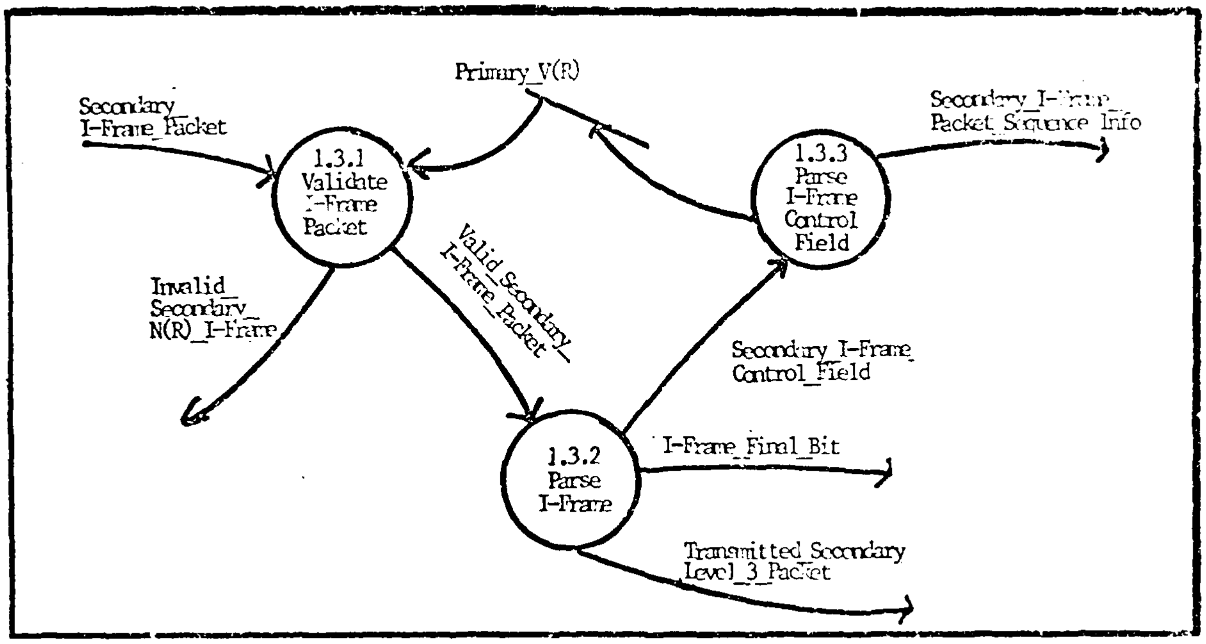
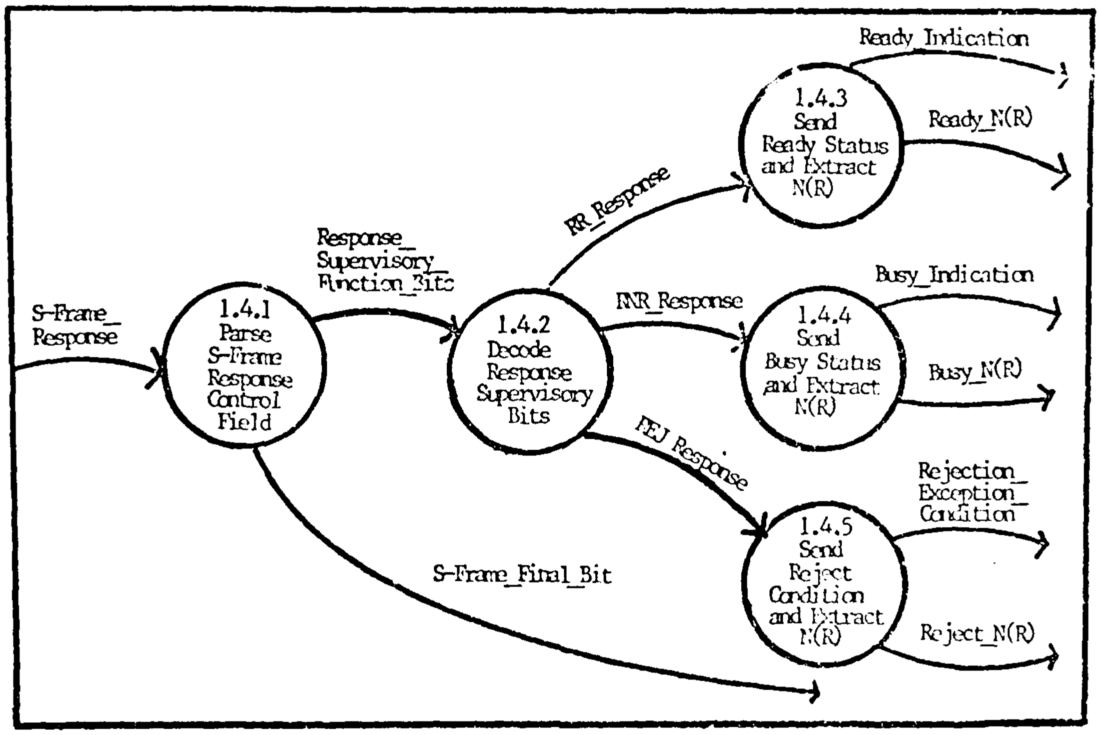


Figure 24 Execute Secondary I-Frame Packet (1.3) DFD



Ready_Indication
1.4.3 Send Ready Status and Extract N(R)
Ready_N(R)
RR_Response
Response_Supervisory_Function_Bits
S-Frame_Response
1.4.1 Parse S-Frame Response Control Field
1.4.2 Decode Response Supervisory Bits
RNR_Response
1.4.4 Send Busy Status and Extract N(R)
Busy_Indication
Busy_N(R)
REJ_Response
1.4.5 Send Reject Condition and Extract N(R)
Rejection_Exception_Condition
Reject_N(R)
S-Frame_Final_Bit

Figure 25 Execute S-Frame Response (1.4) DFD

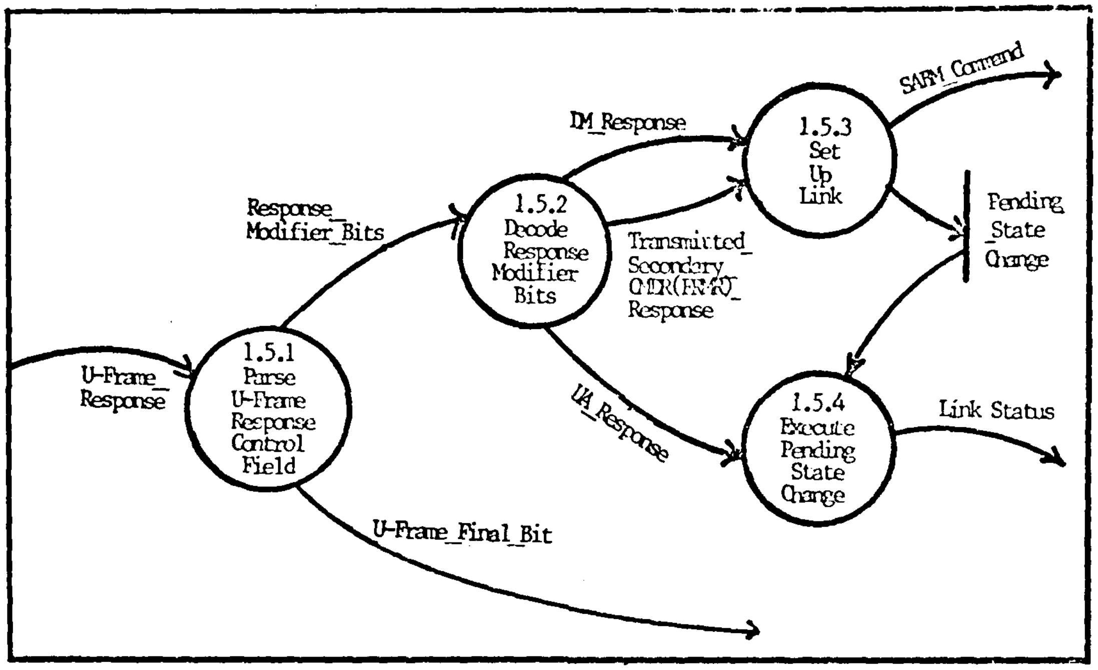


Figure 26 Execute U-Frame Response (1.5) DFD

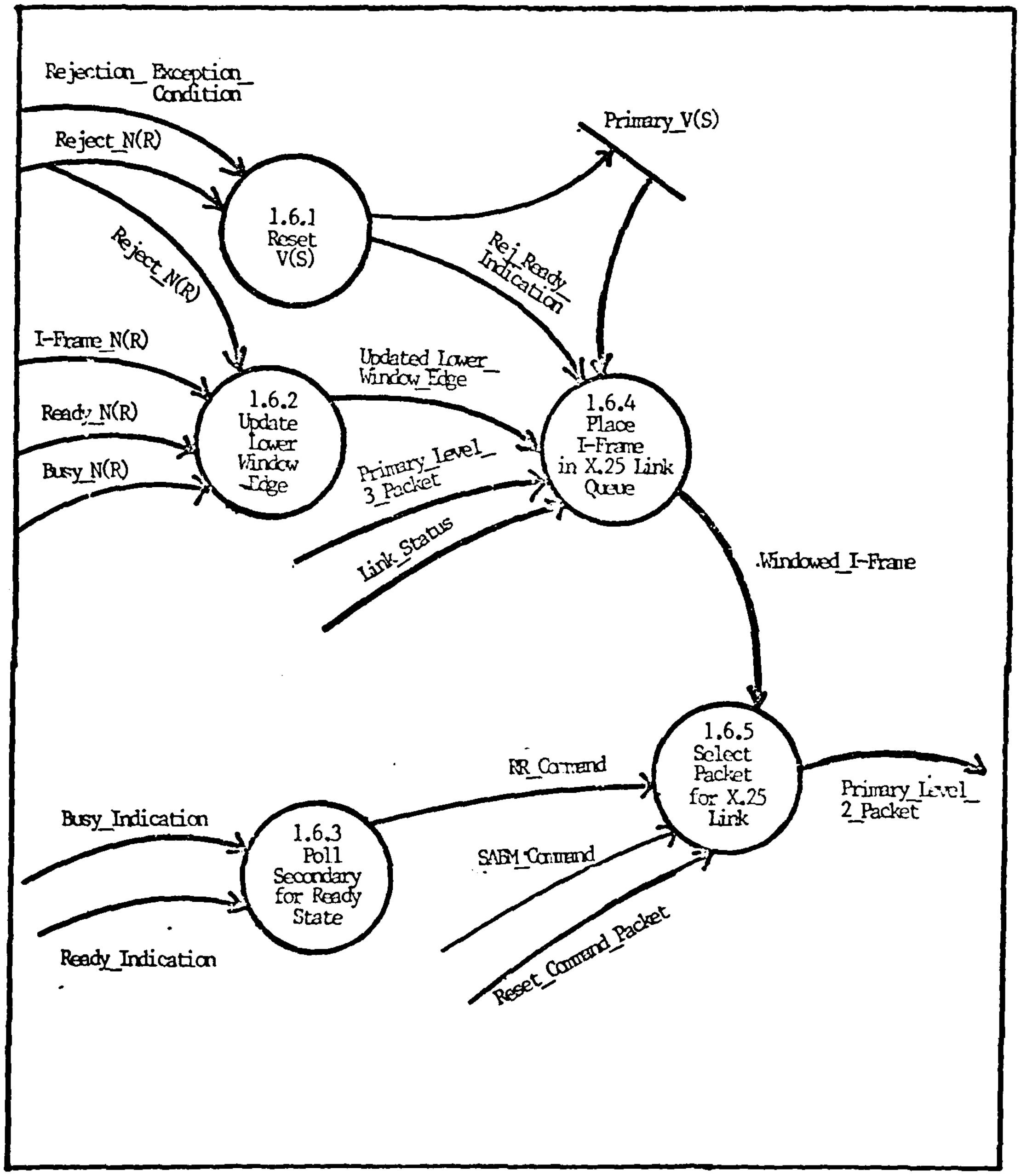


Figure 27 Window Primary Information Blocks (1.6) DFD

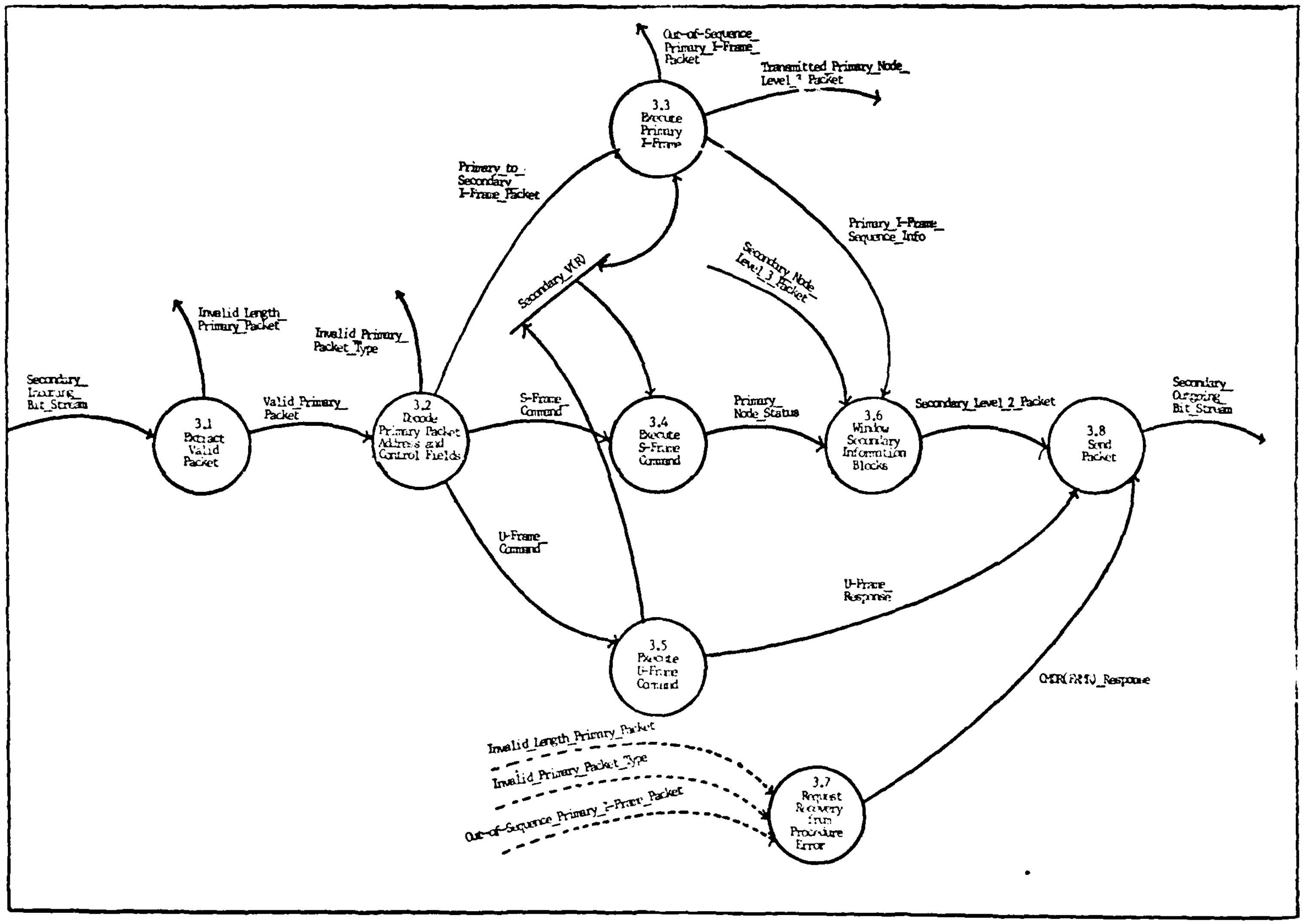
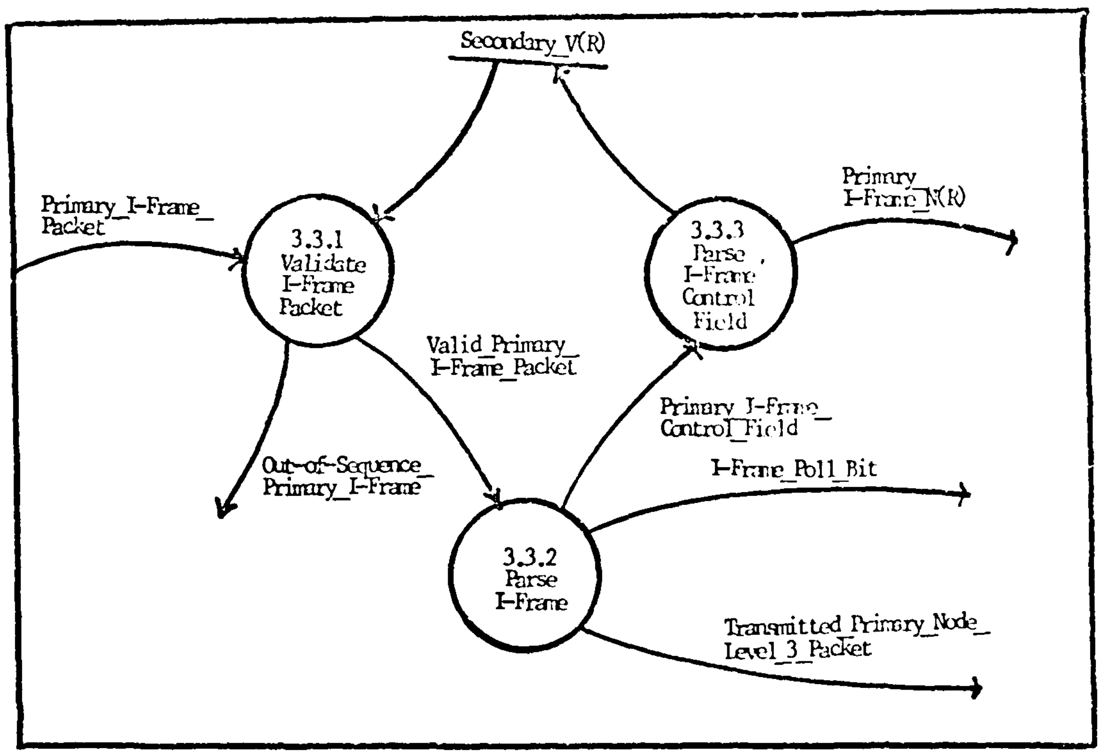

Secondary_Incoming_Bit_Stream
3.1 Extract Valid Packet
Invalid_Length_Primary_Packet
Valid_Primary_Packet
3.2 Decode Primary Packet Address and Control Fields
Invalid_Primary_Packet_Type
Primary_to_Secondary_I-Frame_Packet
3.3 Execute Primary I-Frame
Out-of-Sequence_Primary_I-Frame_Packet
Transmitted_Primary_Node_Level_3_Packet
Secondary_V(R)
Primary_I-Frame_Sequence_Info
Secondary_Node_Level_3_Packet
S-Frame_Command
3.4 Execute S-Frame Command
Primary_Node_Status
3.6 Window Secondary Information Blocks
Secondary_Level_2_Packet
3.8 Send Packet
Secondary_Outgoing_Bit_Stream
U-Frame_Command
3.5 Execute U-Frame Command
U-Frame_Response
CMDR(FRMR)_Response
Invalid_Length_Primary_Packet
Invalid_Primary_Packet_Type
Out-of-Sequence_Primary_I-Frame_Packet
3.7 Request Recovery from Procedure Error

Figure 28. Execute HDLC Protocol at Secondary Node (3) DFD



Secondary_V(R)
Primary_I-Frame_Packet
3.3.1 Validate I-Frame Packet
3.3.3 Parse I-Frame Control Field
Primary I-Frame_N(R)
Valid_Primary_I-Frame_Packet
Primary_I-Frame_Control_Field
Out-of-Sequence_Primary_I-Frame
I-Frame_Poll_Bit
3.3.2 Parse I-Frame
Transmitted_Primary_Node_Level_3_Packet

Figure 29. Execute Primary I-Frame Packet (3.3) DFD

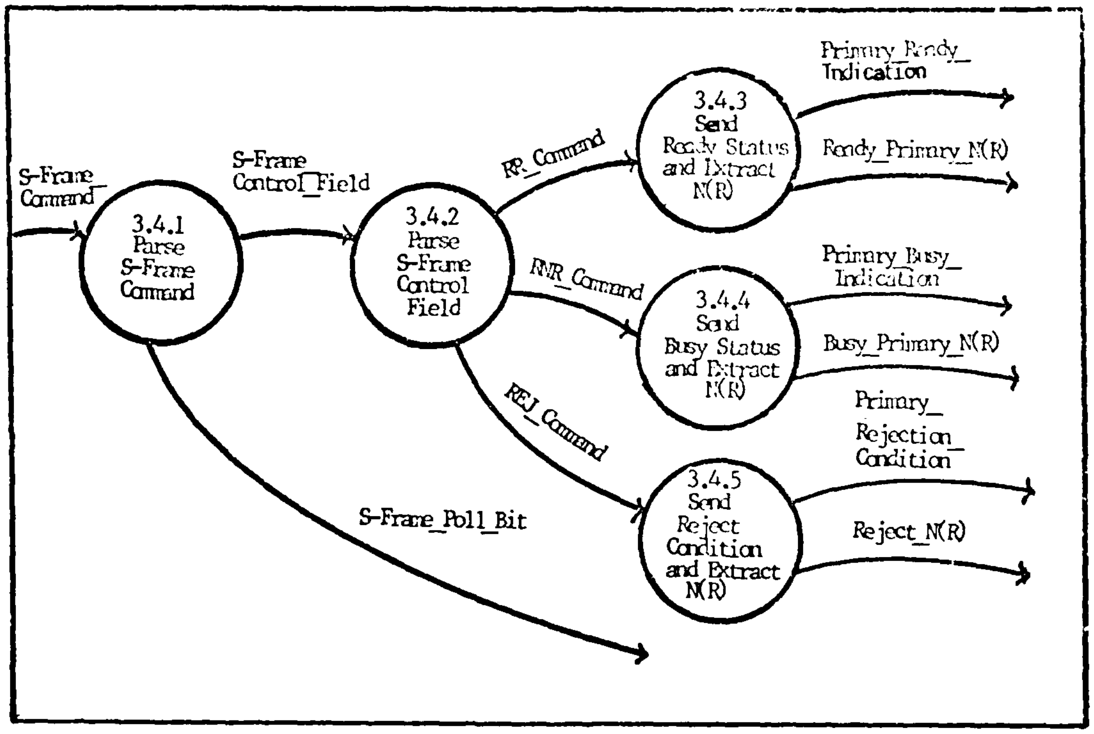

S-Frame_Command
3.4.1 Parse S-Frame Command
S-Frame Control_Field
3.4.2 Parse S-Frame Control Field
RR_Command
RNR_Command
REJ_Command
S-Frame_Poll_Bit
3.4.3 Send Ready Status and Extract N(R)
Primary_Ready_Indication
Ready_Primary_N(R)
3.4.4 Send Busy Status and Extract N(R)
Primary_Busy_Indication
Busy_Primary_N(R)
3.4.5 Send Reject Condition and Extract N(R)
Primary_Rejection_Condition
Reject_N(R)

Figure 30. Execute S-Frame Command (3.4) DFD

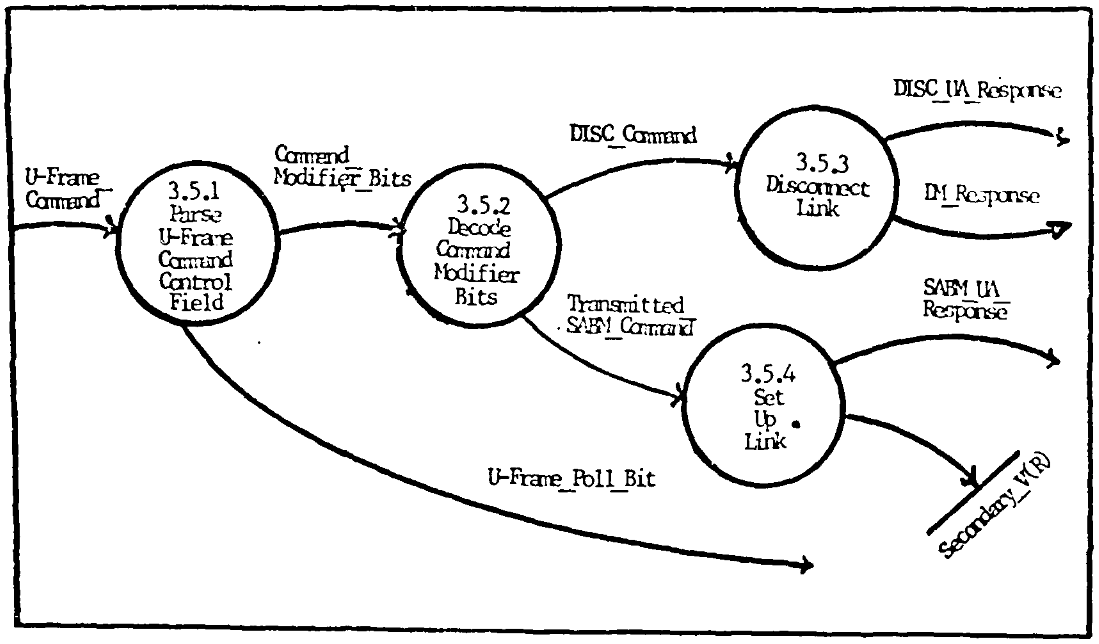


Figure 31. Execute U-Frame Command (3.5) DFD

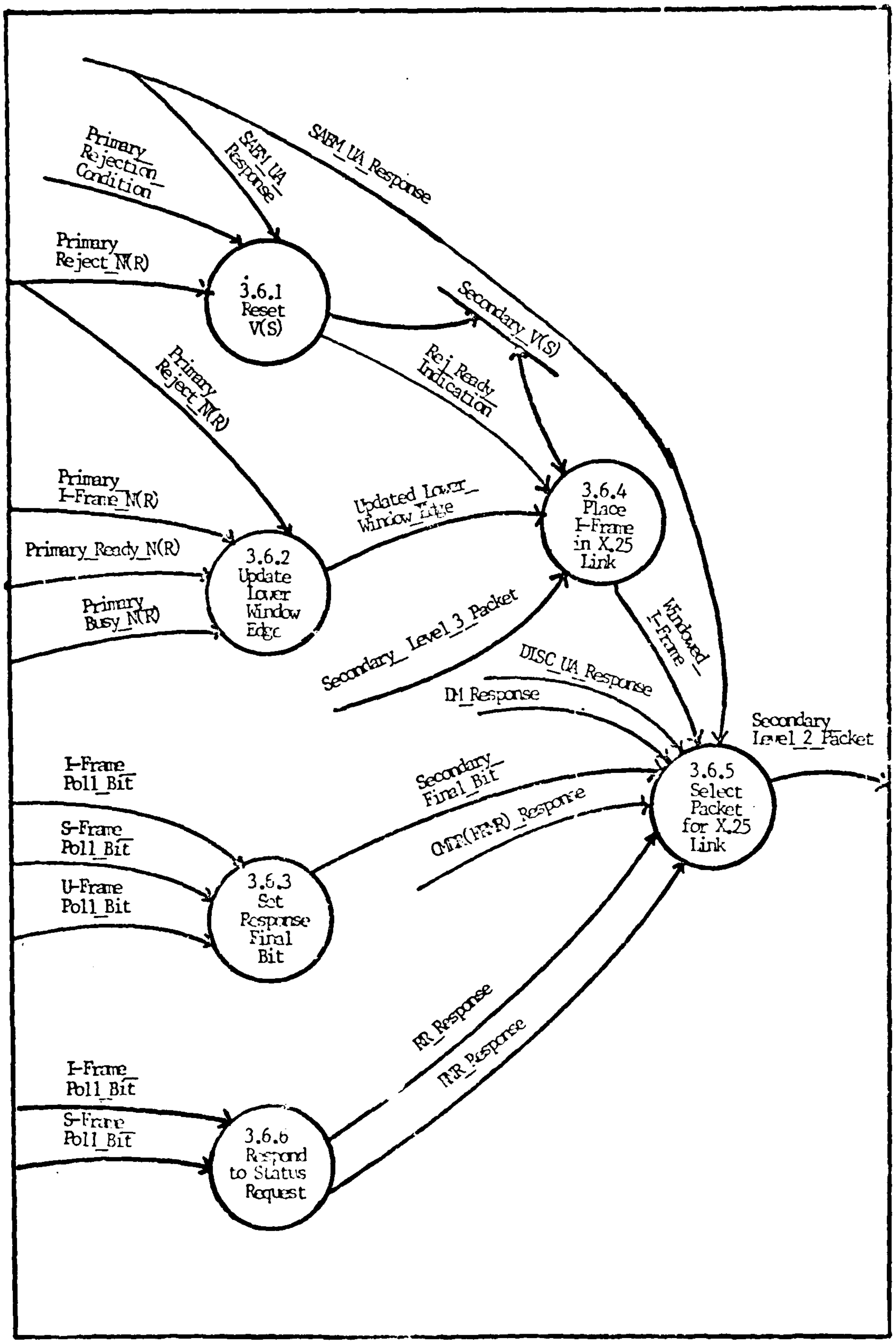


Figure 32. [illegible] Secondary Information Blocks (3.6) DFD

DATA DICTIONARY
FOR HIGH-LEVEL PROTOCOLS

DATA ELEMENT NAME:	ACTIVE-DEVICE-NAME
ALIASES:	NONE
VALUES AND MEANINGS:	
	NAME OF A DEVICE THAT IS ATTACHED TO AN ACTIVE HOST ON THE NETWORK.
NOTES:	NOS LAYER

DATA ELEMENT NAME:	ACTIVE-HOST-NAME
ALIASES:	NONE
VALUES AND MEANINGS:	
	NAME OF A HOST THAT IS ACTIVE ON THE NETWORK.
NOTES:	NOS LAYER

DATA ELEMENT NAME:	ERROR-MESSAGE-FT
ALIASES:	NONE
VALUES AND MEANINGS:	
	ERROR MESSAGE STATING WHY THE FILE CANNOT BE PROPERLY TRANSFERRED
NOTES:	NOS LAYER

DATAFLOW NAME: FILE
ALIASES: NONE
COMPOSITION:

```
FILE = FILE-NAME
       + FILE-TYPE
       + FILE-LENGTH
       + {FILE-CHARACTER}
```

NOTES: APPLICATIONS LAYER

DATAFLOW NAME: FILE-BLOCK
ALIASES: NONE
COMPOSITION:

```
FILE-BLOCK = 1{FILE-CHARACTER}1024
```

NOTES: APPLICATIONS LAYER

DATA ELEMENT NAME:	FILE-CHARACTER
ALIASES:	NONE
VALUES AND MEANINGS:	
	ANY ASCII OR EBCDIC CHARACTER OR 8 BINARY DIGITS
NOTES:	APPLICATIONS LAYER

DATA ELEMENT NAME:	FILE-DEST-DEVICE-NAME
ALIASES:	NONE
VALUES AND MEANINGS:	
	NAME OF THE DEVICE TO WHICH THE FILE IS TO BE TRANSFERRED. NAME MUST CORRESPOND TO ONE OF THOSE IN THE LIST-OF-ACTIVE-HOSTS.
NOTES:	NOS LAYER

DATA ELEMENT NAME:	FILE-DEST-HOST-NAME
ALIASES:	NONE
VALUES AND MEANINGS:	
	NAME OF THE HOST TO WHICH THE FILE IS TO BE TRANSFERRED. NAME MUST CORRESPOND TO ONE OF THOSE IN THE LIST-OF-ACTIVE-HOSTS.
NOTES:	NOS LAYER

DATA ELEMENT NAME:	FILE-LENGTH
ALIASES:	NONE
VALUES AND MEANINGS:	
	A 10-BIT FIELD THAT SPECIFIES THE FILE LENGTH IN FILE-BLOCKS AND THUS ALLOWS FILE-LENGTHS UP TO 1K BLOCKS OR 1 MBYTE
NOTES:	APPLICATIONS LAYER

DATA ELEMENT NAME:	FILE-NAME
ALIASES:	NONE
VALUES AND MEANINGS:	
	ANY CHARACTER STRING CHOSEN BY THE USER TO IDENTIFY THE FILE
NOTES:	NOS LAYER

```
DATA ELEMENT NAME:     FILE-SOURCE-DEVICE-NAME
ALIASES:               NONE
VALUES AND MEANINGS:
                       NAME OF DEVICE THAT CONTAINS THE  FILE  TO
                       BE  TRANSFERRED.   NAME MUST CORRESPOND TO
                       ONE IN THE LIST-OF-ACTIVE-DEVICE-NAMES.
NOTES:                 NOS LAYER


DATA ELEMENT NAME:     FILE-SOURCE-HOST-NAME
ALIASES:               NONE
VALUES AND MEANINGS:
                       NAME OF HOST THAT CONTAINS THE FILE TO  BE
                       TRANSFERRED.   NAME MUST CORRESPOND TO ONE
                       IN THE LIST-OF-ACTIVE-HOSTS.
NOTES:                 NOS LAYER


DATA ELEMENT NAME:     FILE-TRANSFER-PROCEDURE
ALIASES:               NONE
VALUES AND MEANINGS:
                       A SHORT EXPLANATION  OF  HOW  TO  USE  THE
                       NETWORK FILE TRANSFER COMMAND
NOTES:                 NOS LAYER


DATAFLOW NAME:         FILE-TRANSFER-PROCEDURE-REQ
ALIASES:               NONE
COMPOSITION:
                       FILE-TRANSFER-PROCEDURE-REQ =
                                         GENERAL-INFO-REQ
                                         + ",FILE TRANSFER"
NOTES:                 NOS LAYER, APPLICATIONS LAYER


DATAFLOW NAME:         FILE-TRANSFER-MESSAGE
ALIASES:               NONE
COMPOSITION:
                       FILE-TRANSFER-MESSAGE = USER-ID +
                                     [FILE-TRANSFERRED-MESSAGE
                                     |ERROR-MESSAGE-FT]
                       FILE-TRANSFERRED-MESSAGE =
                                      [MODIFIED-FILE-NAME
                                      + FILE-DEST-HOST-NAME
                                      + FILE-DEST-DEVICE-NAME]
NOTES:                 NOS LAYER
```

```
DATA ELEMENT NAME:    FILE-TYPE
ALIASES:              NONE
VALUES AND MEANINGS:
                      A  3-BIT  FIELD  SHOWING  WHETHER  ASCII,
                      EBCDIC, BINARY, ETC
NOTES:                APPLICATIONS LAYER


DATAFLOW NAME:        GENERAL-INFO-REQ
ALIASES:              NONE
COMPOSITION:
                      GENERAL-INFO-REQ = NETWORK-ID + "HELP"
NOTES:                NOS LAYER, APPLICATIONS LAYER


DATA ELEMENT NAME:    GET-COMMAND
ALIASES:              NONE
VALUES AND MEANINGS:
                      THE CHARACTER STRING, "NOS,GET FILE"
NOTES:                APPLICATIONS LAYER


DATAFLOW NAME:        GET-FILE-COMMAND
ALIASES:              NONE
COMPOSITION:
                      GET-FILE-COMMAND = USER-ID
                                  + GET-COMMAND
                                  + FILE-NAME
                                  + FILE-SOURCE-HOST-NAME
                                  + FILE-SOURCE-DEVICE-NAME
                                  + FILE-DEST-HOST-NAME
                                  + FILE-DEST-DEVICE-NAME
NOTES:                APPLICATIONS LAYER


DATA ELEMENT NAME:    GOODBYE
ALIASES:              NONE
VALUES AND MEANINGS: "USER LOGGED OUT OF NETWORK"
NOTES:                NOS LAYER
```

```
DATAFLOW NAME:          HELP-INFORMATION
ALIASES:                NONE
COMPOSITION:
                        HELP-INFORMATION =
                                   [MENU-SELECTION
                                   |FILE-TRANSFER-PROCEDURE
                                   |LIST-OF-ACTIVE-DEVICE-NAMES
                                   |LOGOUT-INFO]
NOTES:                  NOS LAYER


DATAFLOW NAME:          HOST-STRUCTURED-FILE
ALIASES:                NONE
COMPOSITION:
                        HOST-STRUCTURED-FILE = MODIFIED-FILE-NAME
                                             + FILE-TYPE
                                             + FILE-LENGTH
                                             + {FILE-CHARACTER}
NOTES:                  APPLICATIONS LAYER


DATAFLOW NAME:          LIST-OF-ACTIVE-DEVICE-NAMES
ALIASES:                NONE
COMPOSITION:
                        LIST-OF-ACTIVE-DEVICE-NAMES =
                                   {ACTIVE-DEVICE-NAME}
NOTES:                  NOS LAYER


DATAFLOW NAME:          LIST-OF-ACTIVE-DEVICE-NAMES-REQ
ALIASES:                NONE
COMPOSITION:
                        LIST-OF-ACTIVE-DEVICE-NAMES-REQ =
                                      GENERAL-INFO-REQ
                                      + ",LIST ACTIVE DEVICES"
NOTES:                  NOS LAYER, APPLICATIONS LAYER


DATAFLOW NAME:          LIST-OF-ACTIVE-HOSTS
ALIASES:                NONE
COMPOSITION:
                        LIST-OF-ACTIVE-HOSTS = {ACTIVE-HOST-NAME}
NOTES:                  NOS LAYER
```

```
DATAFLOW NAME:          LOCAL-COMMAND
ALIASES:                NONE
COMPOSITION:
                        LOCAL-COMMAND = [VALID-LOCAL-OS-INPUT
                                        |INVALID-INPUT]
NOTES:                  NOS LAYER
                        THE USER IS "LOCAL" TO THE HOST THAT HE
                        DESIGNATES WHEN HE SIGNS ONTO THE NETWORK,
                        NOT NECESSARILY THE ONE TO WHICH THE
                        TERMINAL IS CONNECTED.


DATA ELEMENT NAME:      LOCAL-HOST-NAME
ALIASES:                NONE
VALUES AND MEANINGS:
                        THE NAME OF THE HOST MACHINE TO WHICH THE
                        USER WISHES TO BE "LOCAL"
NOTES:                  NOS-LAYER


DATA ELEMENT NAME:      LOCAL-RESPONSE
ALIASES:                NONE
VALUES AND MEANINGS:
                        LOCAL OPERATING SYSTEM'S RESPONSE TO A
                        LOCAL INPUT
NOTES:                  NOS LAYER


DATAFLOW NAME:          LOGON-COMMAND
ALIASES:                NONE
COMPOSITION:
                        LOGON-COMMAND = NETWORK-ID
                                        + "LOGON"
                                        + (LOCAL-HOST-NAME)
NOTES:                  NOS LAYER


DATAFLOW NAME:          LOGON-MESSAGE
ALIASES:                NONE
COMPOSITION:
                        LOGON-MESSAGE = NETWORK-INTRODUCTION
                                        + LIST-OF-ACTIVE-HOSTS
                                        + (LOCAL-HOST-NAME)
NOTES:                  NOS LAYER
```

DATAFLOW NAME:	LOGOUT-COMMAND
ALIASES:	NONE
COMPOSITION:	
	LOGOUT-COMMAND = NETWORK-ID + "LOGOUT"
NOTES:	NOS LAYER

DATAFLOW NAME:	LOGOUT-INFO-REQ
ALIASES:	NONE
COMPOSITION:	
	LOGOUT-INFO-REQ = GENERAL-INFO-REQ
	+ ",LOGOUT PROCEDURE"
NOTES:	NOS LAYER, APPLICATIONS LAYER

DATAFLOW NAME:	LOGOUT-MESSAGE
ALIASES:	NONE
COMPOSITION:	
	LOGOUT-MESSAGE = GOODBYE
	+ LIST-OF-FILE-TRANSFERS
	LIST-OF-FILE-TRANSFERS =
	{FILE-TRANSFERRED-MESSAGE}
NOTES:	NOS LAYER

DATAFLOW NAME:	MENU-SELECTION
ALIASES:	NONE
COMPOSITION:	
	MENU-SELECTION = USER-ID
	+ NETWORK-COMMAND-SYNTAX
	+ {ACTIVE-HOST-NAME
	+ {ACTIVE-DEVICE-NAME}}
NOTES:	NOS LAYER

DATA ELEMENT NAME:	MODIFIED-FILE-NAME
ALIASES:	NONE
VALUES AND MEANINGS:	
	NAME GIVEN TO THE TRANSFERRED FILE. IT WILL CORRESPOND TO FILE-NAME UNLESS FILE-NAME IS TOO LONG OR CONTAINS ILLEGAL CHARACTERS FOR THE HOST OPERATING SYSTEM TO WHICH THE FILE WAS TRANSFERRED.
NOTES:	NOS LAYER, APPLICATIONS LAYER

DATAFLOW NAME:	NETWORK-COMMAND
ALIASES:	NONE
COMPOSITION:	NETWORK-COMMAND = NETWORK-ID
	+ CHARACTER-STRING
NOTES:	NOS LAYER

DATA ELEMENT NAME:	NETWORK-COMMAND-SYNTAX
ALIASES:	NONE
VALUES AND MEANINGS:	
	SUMMARY OF THE NETWORK COMMANDS AVAILABLE AND THEIR SYNTAXES
NOTES:	NOS LAYER

DATA ELEMENT NAME:	NETWORK-ID
ALIASES:	NONE
VALUES AND MEANINGS:	NETWORK-ID = "NETWORK,"
NOTES:	NOS LAYER

DATA ELEMENT NAME:	NETWORK-INTRODUCTION
ALIASES:	NONE
VALUES AND MEANINGS:	
	A SHORT PARAGRAPH WELCOMING THE USER TO THE NETWORK AND STATING HOW TO GET FURTHER INFORMATION THROUGH THE GENERAL-INFO-REQ.
NOTES:	NOS LAYER

DATAFLOW NAME:	NETWORK-RESPONSE
ALIASES:	NONE
COMPOSITION:	
	NETWORK-RESPONSE = USER-ID
	+ [TRANSMITTED-FILE-TRANSFER-MESSAGE
	|TRANSMITTED-LOGON-MESSAGE
	|TRANSMITTED-LOGOUT-MESSAGE
	|TRANSMITTED-HELP-INFORMATION]
NOTES:	NOS LAYER

DATAFLOW NAME:	NETWORK-STRUCTURED-FILE
ALIASES:	NONE
COMPOSITION:	
	NETWORK-STRUCTURED-FILE = FILE-NAME + FILE-TYPE + FILE-LENGTH + {FILE-BLOCK}
NOTES:	APPLICATIONS LAYER

DATA ELEMENT NAME:	PORT-NUMBER
ALIASES:	NONE
VALUES AND MEANINGS:	
	IF THE HOST HAS MULTIPLE INPUT PORTS, THEN THIS CORRESPONDS TO THE NUMBER OF THE PORT THROUGH WHICH THE USER IS MAKING HIS INPUTS TO THE HOST. IF THE HOST HAS ONLY ONE INPUT PORT, THEN PORT-NUMBER IS ASSIGNED THE VALUE "1".
NOTES:	NOS LAYER, APPLICATIONS LAYER

DATAFLOW NAME:	ROUTED-LOCAL-COMMAND
ALIASES:	NONE
COMPOSITION:	
	ROUTED-LOCAL-COMMAND = LOCAL-COMMAND + USER-ID
NOTES:	NOS LAYER

DATAFLOW NAME:	ROUTED-LOCAL-RESPONSE
ALIASES:	NONE
COMPOSITION:	
	ROUTED-LOCAL-RESPONSE = LOCAL-RESPONSE + USER-ID
NOTES:	NOS LAYER

DATA ELEMENT NAME:	STORE-COMMAND
ALIASES:	NONE
VALUES AND MEANINGS:	
	THE CHARACTER STRING, "NOS,STORE FILE"
NOTES:	APPLICATIONS LAYER

```
DATAFLOW NAME:         STORE-FILE-COMMAND
ALIASES:               NONE
COMPOSITION:
                       STORE-FILE-COMMAND = USER-ID
                                       + STORE-COMMAND
                                       + FILE-DEST-DEVICE-NAME
NOTES:                 APPLICATIONS LAYER


DATA ELEMENT NAME:     TRANSFER-COMMAND
ALIASES:               NONE
VALUES AND MEANINGS:
                       THE CHARACTER STRING, ",TRANSFER FILE"
NOTES:                 NOS LAYER, APPLICATIONS LAYER


DATAFLOW NAME:         TRANSMITTED-FILE-TRANSFER-COMMAND
ALIASES:               NONE
COMPOSITION:
                       TRANSMITTED-FILE-TRANSFER-COMMAND =
                                        TRANSFER-COMMAND
                                        + FILE-TRANSFER-FIELDS
                       FILE-TRANSFER-FIELDS =
                                       FILE-NAME
                                       + FILE-SOURCE-HOST-NAME
                                       + FILE-SOURCE-DEVICE-NAME
                                       + FILE-DEST-HOST-NAME
                                       + FILE-DEST-DEVICE-NAME
NOTES:                 NOS LAYER, APPLICATIONS LAYER


DATAFLOW NAME:         TRANSMITTED-HELP-REQUEST
ALIASES:               NONE
COMPOSITION:
                       TRANSMITTED-HELP-REQUEST = USER-ID
                                        + [GENERAL-INFO-REQ
                                          |SPECIFIC-INFO-REQ]
                       SPECIFIC-INFO-REQ =
                                       [FILE-TRANSFER-INFO-REQ
                                       |LOGOUT-INFO-REQ]
                       FILE-TRANSFER-INFO-REQ =
                                  [FILE-TRANSFER-PROCEDURE-REQ
                                  |LIST-OF-ACTIVE-DEVICE-NAMES-REQ]
NOTES:                 NOS LAYER, APPLICATIONS LAYER
```

```
DATAFLOW NAME:          TRANSMITTED-NETWORK-COMMAND
ALIASES:                NONE
COMPOSITION:
                        TRANSMITTED-NETWORK-COMMAND = USER-ID
                           + [TRANSMITTED-FILE-TRANSFER-COMMAND
                             |TRANSMITTED-SESSION-CONTROL-COMMAND
                             |TRANSMITTED-HELP-REQUEST]
NOTES:                  NOS LAYER


DATAFLOW NAME:          TRANSMITTED-NETWORK-STRUCTURED-FILE
ALIASES:                NONE
COMPOSITION:
                        TRANSMITTED-NETWORK-STRUCTURED-FILE =
                                      USER-ID
                                      + FILE-DEST-HOST-NAME
                                      + FILE-DEST-DEVICE-NAME
                                      + NETWORK-STRUCTURED-FILE
NOTES:                  APPLICATIONS LAYER


DATAFLOW NAME:          TRANSMITTED-SESSION-CONTROL-COMMAND
ALIASES:                NONE
COMPOSITION:            TRANSMITTED-SESSION-CONTROL-COMMAND =
                                           USER-ID
                                           + [LOGON-COMMAND
                                             |LOGOUT-COMMAND]
NOTES:                  NOS LAYER, APPLICATIONS LAYER


DATAFLOW NAME:          USER-COMMAND
ALIASES:                NONE
COMPOSITION:            USER-COMMAND = [LOCAL-COMMAND
                                       |NETWORK-COMMAND]
NOTES:                  NOS LAYER


DATA ELEMENT NAME:      USER-HOST-NAME
ALIASES:                NONE
VALUES AND MEANINGS:
                        NAME OF THE HOST THROUGH WHICH THE USER IS
                        MAKING  HIS  INPUTS  TO THE NETWORK.  NAME
                        MUST   CORRESPOND   TO    ONE    ON    THE
                        LIST-OF-ACTIVE-HOSTS.
NOTES:                  NOS LAYER, APPLICATIONS LAYER
```

DATAFLOW NAME:	USER-ID
ALIASES:	NONE
COMPOSITION:	
	USER-ID = USER-HOST-NAME + PORT-NUMBER
NOTES:	NOS LAYER

DATAFLOW NAME:	USER-RESPONSE
ALIASES:	NONE
COMPOSITION:	
	USER-RESPONSE = [NETWORK-RESPONSE | LOCAL-RESPONSE]
NOTES:	NOS LAYER

DATA ELEMENT NAME:	VALID-LOCAL-OS-INPUT
ALIASES:	NONE
VALUES AND MEANINGS:	
	ANY VALID INPUT FOR THE OPERATING SYSTEM OF THE HOST THAT THE USER IS SIGNED ONTO ON THE NETWORK.
NOTES:	NOS LAYER

FILE DEFINITIONS

```
FILE OR DATABASE NAME: COMMAND-ROUTING-TABLE
ALIASES:               NONE
COMPOSITION:
                       COMMAND-ROUTING-TABLE =
                                    {USER-ID
                                     + LOCAL-HOST-NAME}
ORGANIZATION:          SEQUENTIAL BY USER-ID
NOTES:                 NOS LAYER


FILE OR DATABASE NAME: FILE-TRANSFER-LOG
ALIASES:               NONE
COMPOSITION:
                       FILE-TRANSFER-LOG =
                                 {USER ID
                                  + FILE-TRANSFER-FIELDS}
ORGANIZATION:          PILE
NOTES:                 NOS LAYER


FILE OR DATABASE NAME: NETWORK-CONFIGURATION
ALIASES:               NONE
COMPOSITION:
                       NETWORK-CONFIGURATION =
                                 {ACTIVE-HOST-NAME
                                  + {ACTIVE-DEVICE-NAME}}
ORGANIZATION:
                       ALPHABETICALLY    KEYED    FIRST    ON
                       ACTIVE-HOST-NAME       THEN         ON
                       ACTIVE-DEVICE-NAME
NOTES:                 NOS LAYER
```

PROCESS SPECIFICATIONS

```
PROCESS NAME:         DETERMINE COMMAND TYPE
PROCESS NUMBER:       1.0
PROCESS DESCRIPTION:
If USER-COMMAND contains NETWORK-ID
   then USER-COMMAND = NETWORK-COMMAND
   otherwise USER-COMAND = LOCAL-COMMAND
```

```
PROCESS NAME:         SEND COMMAND TO HOST WITH NOS
PROCESS NUMBER:       2.0
PROCESS DESCRIPTION:
Send NETWORK-COMMAND to NOS-HOST using HOST-TO-HOST-PROTOCOL
   Set SOURCE-FIELD to USER-HOST-NAME
   Set DEST-FIELD to NOS-HOST-NAME
   Set INFO-FIELD to NETWORK-COMMAND
```

```
PROCESS NAME:         DETERMINE NETWORK COMMAND TYPE
PROCESS NUMBER:       3.0
PROCESS DESCRIPTION:
If TRANSMITTED-NETWORK-COMMAND has the following
 COMMAND-FIELD,
   Case 1 TRANSFER-COMMAND:
      then TRANSMITTED-NETWORK-COMMAND =
               TRANSMITTED-FILE-TRANSFER-COMMAND
   Case 2 LOGON-COMMAND or LOGOUT-COMMAND:
      then TRANSMITTED-NETWORK-COMMAND =
               TRANSMITTED-SESSION-CONTROL-COMMAND
   Case 3 GENERAL-INFO-REQ:
      then TRANSMITTED-NETWORK-COMMAND =
               TRANSMITTED-HELP-REQUEST
   otherwise reject TRANSMITTED-NETWORK-COMMAND
```

```
PROCESS NAME:         SEND GET-FILE-COMMAND TO FILE-SOURCE-HOST
PROCESS NUMBER:       4.1
PROCESS DESCRIPTION:
Send GET-FILE-COMMAND to FILE-SOURCE-HOST using
 HOST-TO-HOST-PROTOCOL
   Set SOURCE-FIELD to NOS-HOST
   Set DEST-FIELD to FILE-SOURCE-HOST
   Set INFO-FIELD to GET-FILE-COMMAND
```

```
PROCESS NAME:          GET FILE FROM SOURCE-DEVICE
PROCESS NUMBER:        4.2
PROCESS DESCRIPTION:
Call FILE-FETCH-ROUTINE in the SOURCE-HOST-OS
Use FILE-NAME and FILE-SOURCE-DEVICE-NAME
 from GET-FILE-COMMAND


PROCESS NAME:          RESTRUCTURE FILE FOR NETWORK
PROCESS NUMBER:        4.3
PROCESS DESCRIPTION:
Copy FILE-TYPE into FILE-TYPE-FIELD
Convert FILE-LENGTH to FILE-LENGTH-IN-BYTES and copy into
 FILE-LENGTH-FIELD
Divide FILE into FILE-BLOCKS and fill PARTIAL-LAST-FILE-
 BLOCK with NULLS


PROCESS NAME:          TRANSMIT FILE TO DEST HOST
PROCESS NUMBER:        4.4
PROCESS DESCRIPTION:
Send NETWORK-STRUCTURED-FILE to FILE-DEST-HOST using
 HOST-TO-HOST-PROTOCOL
   While there are FILE-BLOCKS remaining do
      Set SOURCE-FIELD to FILE-SOURCE-HOST-NAME
      Set DEST-FIELD to FILE-DEST-HOST-NAME
      Set INFO-FIELD to FILE-BLOCK


PROCESS NAME:          RESTRUCTURE FILE FOR DEST HOST
PROCESS NUMBER:        4.5
PROCESS DESCRIPTION:
Set FILE-TYPE to FILE-TYPE-FIELD
Set FILE-LENGTH-IN-BYTES to FILE-LENGTH-IN-BYTES-FIELD
Convert FILE-LENGTH-IN-BYTES to FILE-LENGTH
Merge FILE-BLOCKS into HOST-STRUCTURED-FILE


PROCESS NAME:          STORE FILE ON DEST DEVICE
PROCESS NUMBER:        4.6
PROCESS DESCRIPTION:
Call FILE-STORE-ROUTINE in the DEST-HOST-OS
Use FILE-NAME and FILE-DEST-DEVICE-NAME from STORE-COMMAND
 and store HOST-STRUCTURED-FILE
```

```
PROCESS NAME:         DETERMINE SESSION COMMAND TYPE
PROCESS NUMBER:       5.1
PROCESS DESCRIPTION:
If TRANSMITTED-SESSION-CONTROL-COMMAND contains "LOGON"
   then TRANSMITTED-SESSION-CONTROL-COMMAND is a LOGON-COMMAND
   otherwise TRANSMITTED-SESSION-CONTROL-COMMAND is a
    LOGOUT-COMMAND
```

```
PROCESS NAME:         LOG USER ON NETWORK
PROCESS NUMBER:       5.2
PROCESS DESCRIPTION:
If LOCAL-HOST-NAME is included in the LOGON-COMMAND
   then enter LOCAL-HOST-NAME into the COMMAND-ROUTING-TABLE
    using the USER-ID in the LOGON-COMMAND
   otherwise access NETWORK-CONFIGURATION-TABLE using
    USER-ID and set COMMAND-ROUTING-TABLE to USER-HOST-NAME
Access NETWORK-CONFIGURATION-TABLE and get LIST-OF-ACTIVE-
 HOSTS
Output LOGON-MESSAGE
```

```
PROCESS NAME:         LOG USER OFF NETWORK
PROCESS NUMBER:       5.3
PROCESS DESCRIPTION:
Reset LOCAL-HOST-NAME in COMMAND-ROUTING-TABLE to USER-HOST-NAME
Get LIST-OF-FILE-TRANSFER-MESSAGES from FILE-TRANSFER-LOG
Output LOGOUT-MESSAGE
```

```
PROCESS NAME:         DETERMINE HELP-INFORMATION REQUESTED
PROCESS NUMBER:       6.1
PROCESS DESCRIPTION:
If TRANSMITTED-HELP-REQUEST contains the HELP-FIELD
   Case 1  "FILE TRANSFER":
     Then TRANSMITTED-HELP-REQUEST is a FILE-TRANSFER-
      PROCEDURE-REQ
   Case 2  "LIST ACTIVE DEVICES":
     Then TRANSMITTED-HELP-REQUEST is a LIST-ACTIVE-
      DEVICE-NAMES-REQ
   Case 3  "LOGOUT PROCEDURE":
     Then TRANSMITTED-HELP-REQUEST is a LOGOUT-INFO-REQ
   Otherwise, TRANSMITTED-HELP-REQUEST is a GENERAL-
    INFO-REQ
```

```
PROCESS NAME:          PROVIDE GENERAL NETWORK INTRODUCTION
PROCESS NUMBER:        6.2
PROCESS DESCRIPTION:
Output MENU-SELECTION and USER-ID


PROCESS NAME:          PROVIDE PROCEDURE FOR TRANSFERRING FILES
PROCESS NUMBER:        6.3
PROCESS DESCRIPTION:
Output FILE-TRANSFER-PROCEDURE and USER-ID


PROCESS NAME:          PROVIDE LIST-OF-ACTIVE-DEVICE-NAMES
PROCESS NUMBER:        6.4
PROCESS DESCRIPTION:
Access NETWORK-CONFIGURATION-TABLE
Output LIST-OF-ACTIVE-DEVICE-NAMES and USER-ID


PROCESS NAME:          PROVIDE PROCEDURE FOR LOGGING OFF NETWORK
PROCESS NUMBER:        6.5
PROCESS DESCRIPTION:
Output LOGOUT-INFO and USER-ID


PROCESS NAME:          SEND MESSAGE TO USER-HOST
PROCESS NUMBER:        7.0
PROCESS DESCRIPTION:
Send NETWORK-RESPONSE to USER-HOST using HOST-TO-HOST-
 PROTOCOL
   Set SOURCE-FIELD to FILE-DEST-HOST
   Set DEST-FIELD to USER-HOST-NAME
   Set INFO-FIELD to NETWORK-RESPONSE


PROCESS NAME:          ROUTE LOCAL COMMAND
PROCESS NUMBER:        8.0
PROCESS DESCRIPTION:
Find LOCAL-HOST-NAME for USER from COMMAND-ROUTING-TABLE
Use HOST-TO-HOST-PROTOCOL to send LOCAL-COMMAND to
 LOCAL-HOST
   Set SOURCE-FIELD to USER-HOST-NAME
   Set DEST-FIELD to LOCAL-HOST-NAME
   Set INFO-FIELD to LOCAL-COMMAND
```

```
PROCESS NAME:          EXECUTE ROUTED-LOCAL-COMMAND
PROCESS NUMBER:        9.0
PROCESS DESCRIPTION:
Use LOCAL-OPERATING-SYSTEM to execute ROUTED-LOCAL-COMMAND



PROCESS NAME:          ROUTE LOCAL RESPONSE
PROCESS NUMBER:        10.0
PROCESS DESCRIPTION:
Send LOCAL-RESPONSE to USER-HOST using HOST-TO-HOST-
 PROTOCOL
   Set SOURCE-FIELD to LOCAL-HOST-NAME
   Set DEST-FIELD to USER-HOST-NAME
   Set INFO-FIELD to LOCAL-RESPONSE
```

AD-A100 822 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN OF A LOCAL COMPUTER NETWORK FOR THE AIR FORCE INSTITUTE --ETC(U)
MAR 81 W C HOBART
UNCLASSIFIED AFIT/GE/EE/81M-3 NL

3 OF 4
AD A
100822

DATA DICTIONARY

FOR X.25 LEVEL 3 PROTOCOL

DATA ELEMENT NAME:	CALL-CONNECTED-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (00001111)
NOTES:	CALLING NODE AND HOST PROTOCOL LAYERS

DATAFLOW NAME:	CALL-CONNECTION-PACKET
ALIASES:	NONE
COMPOSITION:	CALL-CONNECTION-PACKET = [CALL-CONNECTED-PACKET | CALLING-NODE-CLEAR-INDICATION-PACKET]
NOTES:	CALLING NODE PROTOCOL LAYER

DATA ELEMENT NAME:	CALLED-HOST-INTERRUPT-CONFIRMATION-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 1 0 0 1 1 1)
NOTES:	CALLED HOST AND NODE PROTOCOL LAYER

DATAFLOW NAME:	CALLED-HOST-LEVEL-3-PACKET
ALIASES:	CALLED-HOST-TO-CALLED-NODE-LEVEL-3-PACKET
COMPOSITION:	SEE ALIAS
NOTES:	SEE ALIAS

DATAFLOW NAME:	CALLED-HOST-SUPERVISORY-PACKET
ALIASES:	NONE
COMPOSITION:	CALLED-HOST-SUPERVISORY-PACKET = [HOST-RR | HOST-RNR | HOST-REJ]
NOTES:	CALLED NODE PROTOCOL LAYER

DATAFLOW NAME:	CALLED-HOST-TO-CALLED-NODE-LEVEL-3-PACKET
ALIASES:	CALLED-HOST-LEVEL-3-PACKET
COMPOSITION:	CALLED-HOST-TO-CALLED-NODE-LEVEL-3-PACKET= [CALL-ACCEPTED-PACKET | WINDOWED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET | CALLED-HOST-INTERRUPT-PACKET | CALLED-HOST-CLEAR-REQUEST | CALLED-HOST-RESET-REQUEST | CALLED-HOST-RESTART-REQUEST | CALLED-HOST-SUPERVISORY-PACKET]
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	CALLED-HOST-TO-CALLING-HOST-BUFFERED-SEQUENCE
ALIASES:	NONE
COMPOSITION:	CALLED-HOST-TO-CALLING-HOST-BUFFERED-SEQUENCE = {CALLED-HOST-TO-CALLING-HOST-DATA-PACKET}
NOTES:	CALLING HOST PROTOCOL LAYER

DATA ELEMENT NAME:	CALLED-HOST-TO-CALLING-HOST-DATA
ALIASES:	NONE
VALUES AND MEANINGS:	ANY STRING OF BITS NEEDING TO BE TRANSMITTED FROM THE CALLED HOST TO THE CALLING HOST
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
ALIASES:	NONE
COMPOSITION:	CALLED-HOST-TO-CALLING-HOST-DATA-PACKET = [CALLED-HOST-TO-CALLING-HOST-CATEGORY-1-PACKET | CALLED-HOST-TO-CALLING-HOST-CATEGORY-2-PACKET]
NOTES:	CALLED HOST PROTOCOL LAYER

DATA ELEMENT NAME:	CALLED-HOST-TO-CALLING-HOST-INTERRUPT-DATA
ALIASES:	NONE
VALUES AND MEANINGS:	ANY DATA THAT MUST BE TRANSMITTED FROM THE CALLED HOST TO THE CALLING HOST WITHOUT USING THE LEVEL 3 FLOW CONTROL THE DATA MUST NOT BE MORE THAN 8 BITS IN LENGTH
NOTES:	CALLED HOST PROTOCOL LAYER

DATAFLOW NAME:	CALLED-HOST-TO-CALLING-HOST-PACKET
ALIASES:	NONE
COMPOSITION:	CALLED-HOST-TO-CALLING-HOST-PACKET = [CALLING-NODE-INTERRUPT-PACKET | CALL-CONNECTED-PACKET | CALLING-NODE-WINDOWED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET]
NOTES:	CALLING NODE PROTOCOL LAYER

DATA ELEMENT NAME:	CALLED-NODE-CLEAR-CONFIRMATION-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 1 0 1 1 1)
NOTES:	CALLED NODE AND HOST PROTOCOL LAYERS

DATAFLOW NAME:	CALLED-NODE-CONFIRMATION-PACKET
ALIASES:	NONE
COMPOSITION:	CALLED-NODE-CONFIRMATION-PACKET = [CALLED-NODE-INTERRUPT-CONFIRMATION-PACKET | CALLED-NODE-CLEAR-CONFIRMATION-PACKET | CALLED-NODE-RESET-CONFIRMATION-PACKET | CALLED-NODE-RESTART-CONFIRMATION-PACKET]
NOTES:	CALLED NODE PROTOCOL LAYER

DATA ELEMENT NAME:	CALLED-NODE-INTERRUPT-CONFIRMATION-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 1 0 0 1 1 1)
NOTES:	CALLED NODE AND HOST PROTOCOL LAYERS

DATA ELEMENT NAME:	CALLED-NODE-INTERRUPT-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	CONTAINS UP TO 8 BITS OF DATA FROM THE CALLING HOST TO THE CALLED HOST THAT DOES NOT HAVE TO BE TRANSMITTED USING THE LEVEL 3 FLOW CONTROL
NOTES:	CALLED NODE PROTOCOL LAYER

DATAFLOW NAME:	CALLED-NODE-LEVEL-3-PACKET
ALIASES:	CALLED-NODE-TO-CALLING-NODE-LEVEL-3-PACKET
COMPOSITION:	SEE ALIAS
NOTES:	SEE ALIAS

DATA ELEMENT NAME:	CALLED-NODE-RESET-CONFIRMATION-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 1 1 1 1 1)
NOTES:	CALLED HOST AND NODE PROTOCOL LAYERS

DATA ELEMENT NAME:	CALLED-NODE-RESTART-CONFIRMATION-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (1 1 1 1 1 1 1 1)
NOTES:	CALLED NODE AND HOST PROTOCOL LAYERS

DATAFLOW NAME:	CALLED-NODE-SUPERVISORY-PACKET
ALIASES:	NONE
COMPOSITION:	CALLED-NODE-SUPERVISORY-PACKET = [NODE-RR | NODE-RNR | NODE-REJ] + P(R)
NOTES:	CALLED NODE AND HOST PROTOCOLS

DATAFLOW NAME:	CALLED-NODE-TO-CALLED-HOST-LEVEL-3-PACKET
ALIASES:	NONE
COMPOSITION:	CALLED-NODE-TO-CALLED-HOST-LEVEL-3-PACKET= [NODE-CALL-MAINTENANCE-PACKET | CALLED-NODE-SUPERVISORY-PACKET | TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET | TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-NODE-INTERRUPT-PACKET]
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	CALLED-NODE-TO-CALLING-NODE-LEVEL-3-PACKET
ALIASES:	CALLED-NODE-LEVEL-3-PACKET
COMPOSITION:	CALLED-NODE-TO-CALLING-NODE-LEVEL-3-PACKET= [CALLED-NODE-INTERRUPT-PACKET | CALL-ACCEPTED-PACKET | CALLED-HOST-TO-CALLING-HOST-DATA-PACKET]
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	CALLED-NODE-WINDOWED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
ALIASES:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
COMPOSITION:	SEE ALIAS
NOTES:	SEE ALIAS

DATAFLOW NAME:	CALLING-HOST-CONFIRMATION-PACKET
ALIASES:	CALLING-HOST-INTERRUPT-CONFIRMATION-PACKET
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	SEE ALIAS

DATA ELEMENT NAME:	CALLING-HOST-INTERRUPT-CONFIRMATION-PACKET
ALIASES:	CALLING-HOST-CONFIRMATION-PACKET
VALUES AND MEANINGS:	OCTET 3 = (00100111)
NOTES:	CALLING, CALLED HOST AND NODE PROTOCOL LAYERS

DATAFLOW NAME:	CALLING-HOST-LEVEL-3-PACKET
ALIASES:	CALLING-HOST-TO-CALLING-NODE-LEVEL-3-PACKET
COMPOSITION:	CALLING-HOST-LEVEL-3-PACKET= [HOST-CALL-SETUP-PACKET | WINDOWED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET | CALLING-HOST-INTERRUPT-CONFIRMATION-PACKET]
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	CALLING-HOST-SUPERVISORY-PACKET
ALIASES:	NONE
COMPOSITION:	CALLING-HOST-SUPERVISORY-PACKET = [HOST-RR | HOST-RNR | HOST-REJ]
NOTES:	CALLING NODE PROTOCOL LAYER

DATAFLOW NAME:	CALLING-HOST-TO-CALLED-HOST-BUFFERED-SEQUENCE
ALIASES:	NONE
COMPOSITION:	CALLING-HOST-TO-CALLED-HOST-BUFFERED-SEQUENCE = {CALLED-HOST-TO-CALLING-HOST-DATA-PACKET}
NOTES:	CALLED HOST PROTOCOL LAYER

DATA ELEMENT NAME:	CALLING-HOST-TO-CALLED-HOST-DATA
ALIASES:	NONE
VALUES AND MEANINGS:	ANY STRING OF BITS NEEDING TO BE TRANSMITTED FROM THE CALLING HOST TO THE CALLED HOST
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
ALIASES:	NONE
COMPOSITION:	CALLING-HOST-TO-CALLED-HOST-DATA-PACKET = [CALLING-HOST-TO-CALLED-HOST-CATEGORY-1-PACKET | CALLING-HOST-TO-CALLED-HOST-CATEGORY-2-PACKET]
NOTES:	CALLING HOST PROTOCOL LAYER

DATA ELEMENT NAME:	CALLING-HOST-TO-CALLED-HOST-INTERRUPT-DATA
ALIASES:	NONE
VALUES AND MEANINGS:	ANY DATA THAT MUST BE TRANSMITTED BETWEEN HOSTS WITHOUT USING THE FLOW CONTROL MECHANISM. DATA LENGTH MUST BE LESS THAN OR EQUAL TO 8 BITS.
NOTES:	CALLING HOST PROTOCOL LAYER

DATAFLOW NAME:	CALLING-HOST-TO-CALLED-HOST-PACKET
ALIASES:	NONE
COMPOSITION:	CALLING-HOST-TO-CALLED-HOST-PACKET = [CALLED-NODE-INTERRUPT-PACKET | INCOMING-CALL-PACKET | CALLED-NODE-WINDOWED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET | CALLED-NODE-SUPERVISORY-PACKET]
NOTES:	CALLED NODE PROTOCOL LAYER

DATAFLOW NAME:	CALLING-HOST-TO-CALLING-NODE-LEVEL-3-PACKET
ALIASES:	CALLING-HOST-LEVEL-3-PACKET
COMPOSITION:	SEE ALIAS
NOTES:	SEE ALIAS

DATA ELEMENT NAME:	CALLING-NODE-CLEAR-CONFIRMATION-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 1 0 1 1 1)
NOTES:	CALLING NODE AND HOST PROTOCOL LAYERS

DATA ELEMENT NAME:	CALLING-NODE-CLEAR-INDICATION-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (00010011)
NOTES:	CALLING NODE PROTOCOL LAYER

DATAFLOW NAME:	CALLING-NODE-CONFIRMATION-PACKET
ALIASES:	NONE
COMPOSITION:	CALLING-NODE-CONFIRMATION-PACKET = [CALLING-NODE-INTERRUPT-CONFIRMATION-PACKET | CALLING-NODE-CLEAR-CONFIRMATION-PACKET | CALLING-NODE-RESET-CONFIRMATION-PACKET | CALLING-NODE-RESTART-CONFIRMATION-PACKET]
NOTES:	CALLING NODE PROTOCOL LAYER

DATA ELEMENT NAME:	CALLING-NODE-INTERRUPT-CONFIRMATION-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 1 0 0 1 1 1)
NOTES:	CALLING HOST AND NODE PROTOCOL LAYERS

DATA ELEMENT NAME:	CALLING-NODE-INTERRUPT-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	CONTAINS UP TO 8 BITS OF DATA FROM THE CALLED HOST TO THE CALLING HOST THAT DOES NOT HAVE TO BE TRANSMITTED USING THE LEVEL 3 FLOW CONTROL
NOTES:	CALLING NODE PROTOCOL LAYER

DATAFLOW NAME:	CALLING-NODE-LEVEL-3-PACKET
ALIASES:	CALLING-NODE-TO-CALLED-NODE-LEVEL-3-PACKET
COMPOSITION:	SEE ALIAS
NOTES:	SEE ALIAS

DATA ELEMENT NAME:	CALLING-NODE-RESET-CONFIRMATION-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 1 1 1 1 1)
NOTES:	CALLING NODE AND HOST PROTOCOL LAYERS

DATA ELEMENT NAME:	CALLING-NODE-RESTART-CONFIRMATION-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (1 1 1 1 1 1 1 1)
NOTES:	CALLING HOST AND NODE PROTOCOL LAYERS

DATAFLOW NAME:	CALLING-NODE-SUPERVISORY-PACKET
ALIASES:	NONE
COMPOSITION:	CALLING-NODE-SUPERVISORY-PACKET = [NODE-RR | NODE-RNR | NODE-REJ] + P(R)
NOTES:	CALLING HOST PROTOCOL LAYER

DATAFLOW NAME:	CALLING-NODE-TO-CALLED-NODE-LEVEL-3-PACKET
ALIASES:	CALLING-NODE-LEVEL-3-PACKET
COMPOSITION:	CALLING-NODE-TO-CALLED-NODE-LEVEL-3-PACKET= [CALLING-NODE-INTERRUPT-PACKET | INCOMING-CALL-PACKET | CALLING-HOST-TO-CALLED-HOST-DATA-PACKET]
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	CALLING-NODE-TO-CALLED-NODE-PACKET
ALIASES:	NONE
COMPOSITION:	CALLING-NODE-TO-CALLED-NODE-PACKET= [CALLING-NODE-INTERRUPT-PACKET | INCOMING-CALL-PACKET | CALLING-HOST-TO-CALLED-HOST-DATA-PACKET]
NOTES:	CALLING NODE PROTOCOL LAYER

DATAFLOW NAME:	CALLING-NODE-TO-CALLING-HOST-LEVEL-3-PACKET
ALIASES:	NONE
COMPOSITION:	CALLING-NODE-TO-CALLING-HOST-LEVEL-3-PACKET [CALLING-NODE-CONFIRMATION-PACKET | CALLED-HOST-CALLING-HOST-PACKET | ERROR-RECOVERY-PACKET]
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	CALLING-NODE-WINDOWED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
ALIASES:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
COMPOSITION:	SEE ALIAS
NOTES:	SEE ALIAS

DATAFLOW NAME:	DATA
ALIASES:	NONE
COMPOSITION:	DATA=[CALLING-HOST-TO-CALLED-HOST-DATA |CALLED-HOST-TO-CALLING-HOST-DATA]
NOTES:	CONTEXT LAYER

DATA ELEMENT NAME:	ERROR-RECOVERY-PACKET
ALIASES:	RESTART-INDICATION-PACKET
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	SEE ALIAS

DATAFLOW NAME:	HOST-CALL-MAINTENANCE-PACKET
ALIASES:	NONE
COMPOSITION:	HOST-CALL-MAINTENANCE-PACKET = [CALL-ACCEPTED-PACKET | RESET-REQUEST-PACKET | RESTART-REQUEST-PACKET | CLEAR-REQUEST-PACKET]
NOTES:	CALLED HOST PROTOCOL LAYER

DATAFLOW NAME:	HOST-CALL-SETUP-PACKET
ALIASES:	NONE
COMPOSITION:	HOST-CALL-SETUP-PACKET = [CALL-REQUEST-PACKET | RESET-REQUEST-PACKET | RESTART-REQUEST-PACKET | CLEAR-REQUEST-PACKET]
NOTES:	CALLING HOST PROTOCOL LAYER

DATA ELEMENT NAME:	HOST-REJ
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (XXX01001)
NOTES:	CALLING, CALLED HOST PROTOCOL LAYERS

DATA ELEMENT NAME:	HOST-RNR
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (XXX00101)
NOTES:	CALLING, CALLED HOST PROTOCOL LAYERS

DATA ELEMENT NAME:	HOST-RR
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (XXX00001)
NOTES:	CALLING, CALLED HOST PROTOCOL LAYERS

DATA ELEMENT NAME:	INCOMING-CALL-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 0 1 0 1 1)
NOTES:	CALLING, CALLED NODE PROTOCOL LAYERS

DATA ELEMENT NAME:	LAST-PACKET-BIT
ALIASES:	NONE
VALUES AND MEANINGS:	BIT 5 OF OCTET 3 IS SET TO 0
NOTES:	CALLING, CALLED HOST PROTOCOL LAYERS

DATA ELEMENT NAME:	LOCAL-PROCEDURE-ERROR
ALIASES:	NONE
VALUES AND MEANINGS:	AN ERROR CAN BE CAUSED BY AN INVALID CALLING-HOST-LEVEL-3-PACKET OR AN INVALID CALLED-NODE-LEVEL-3-PACKET ALSO, ERRORS CAN OCCUR WHEN THE PACKET RECEIVED IS ILLEGAL FOR THE CURRENT STATE OF THE CALLING NODE
NOTES:	CALLING NODE PRTOCOL LAYER

DATA ELEMENT NAME:	MORE-DATA-BIT
ALIASES:	NONE
VALUES AND MEANINGS:	BIT 5 OF OCTET 3 IS SET TO 1
NOTES:	CALLING, CALLED HOST PROTOCOL LAYERS

DATAFLOW NAME:	NODE-CALL-MAINTENANCE-PACKET
ALIASES:	NONE
COMPOSITION:	NODE-CALL-MAINTENANCE-PACKET = [RELAYED-INCOMING-CALL-PACKET | CALLED-NODE-CLEAR-INDICATION-PACKET| CALLED-NODE-RESET-INDICATION-PACKET| CALLED-NODE-RESTART-INDICATION-PACKET| CALLED-NODE-CLEAR-CONFIRMATION-PACKET| CALLED-NODE-RESET-CONFIRMATION-PACKET| CALLED-NODE-RESTART-CONFIRMATION-PACKET | CALLED-NODE-INTERRUPT-CONFIRMATION-PACKET]
NOTES:	CALLED HOST PROTOCOL LAYER

DATAFLOW NAME:	NODE-CALL-SETUP-PACKET
ALIASES:	NONE
COMPOSITION:	NODE-CALL-SETUP-PACKET = [CALL-CONNECTED-PACKET | NODE-CLEAR-INDICATION-PACKET | CALLING-NODE-CLEAR-CONFIRMATION-PACKET | CALLING-NODE-RESET-CONFIRMATION-PACKET | CALLING-NODE-RESTART-CONFIRMATION-PACKET | CALLING-NODE-RESET-INDICATION-PACKET | CALLING-NODE-RESTART-INDICATION-PACKET]
NOTES:	CALLING HOST PROTOCOL LAYER

DATA ELEMENT NAME:	NODE-INTERRUPT-IDENTIFIER
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (00100011)
NOTES:	CALLING, CALLED HOST PROTOCOL LAYERS

DATA ELEMENT NAME:	PROCEDURE ERROR
ALIASES:	NONE
VALUES AND MEANINGS:	AN ERROR CAN BE CAUSED BY AN INVALID CALLED-HOST-LEVEL-3-PACKET OR AN INVALID CALLING-NODE-LEVEL-3-PACKET. ALSO, ERRORS CAN OCCUR WHEN THE PACKET RECEIVED IS ILLEGAL FOR THE CURRENT STATE OF THE CALLED NODE.
NOTES:	CALLED NODE PROTOCOL LAYER

DATA ELEMENT NAME:	P(R)
ALIASES:	PACKET-RECEIVE-NUMBER
VALUES AND MEANINGS:	BITS 8,7,6 OF OCTET 3 IN DATA AND SUPERVISORY PACKETS
NOTES:	CALLING, CALLED NODE AND HOST PROTOCOLS

DATA ELEMENT NAME:	P(S)
ALIASES:	PACKET-SEND-NUMBER
VALUES AND MEANINGS:	BITS 4,3,2 OF OCTET 3 IN DATA AND SUPERVISORY PACKETS
NOTES:	CALLING, CALLED NODE AND HOST PROTOCOLS

DATA ELEMENT NAME:	QUEUED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	THIS IS A CALLED-HOST-TO-CALLING-HOST-DATA-PACKET THAT HAS BEEN PLACED IN THE TRANSMISSION QUEUE TO BE SENT TO THE CALLED NODE
NOTES:	CALLED HOST PROTOCOL LAYER

DATA ELEMENT NAME:	QUEUED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	THIS IS A CALLING-HOST-TO-CALLED-HOST-DATA-PACKET THAT HAS BEEN PLACED IN THE TRANSMISSION QUEUE TO BE SENT TO THE CALLING NODE
NOTES:	CALLING HOST PROTOCOL LAYER

DATA ELEMENT NAME:	RECEIVED-CALL-ACCEPTED-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (00001111)
NOTES:	CALLING, CALLED NODE PROTOCOL LAYERS

DATA ELEMENT NAME:	RECEIVED-CALL-REQUEST-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 0 1 0 1 1)
NOTES:	CALLING HOST AND NODE PROTOCOL LAYERS

DATA ELEMENT NAME:	RECEIVED-CALLED-HOST-CLEAR-REQUEST
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 1 0 0 1 1)
NOTES:	CALLED NODE AND HOST PROTOCOL LAYERS

DATA ELEMENT NAME:	RECEIVED-CALLED-HOST-INTERRUPT-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	ANY STRING OF UP TO 8 BITS THAT IS TRANSMITTED FROM THE CALLED HOST TO THE CALLING HOST WITHOUT USING LEVEL 3 FLOW CONTROL
NOTES:	CALLING NODE PROTOCOL LAYER

DATA ELEMENT NAME:	RECEIVED-CALLED-HOST-RESET-REQUEST
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 1 1 0 1 1)
NOTES:	CALLED NODE AND HOST PROTOCOL LAYERS

DATA ELEMENT NAME:	RECEIVED-CALLED-HOST-RESTART-REQUEST
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (1 1 1 1 1 0 1 1)
NOTES:	CALLED HOST AND NODE PROTOCOL LAYERS

DATA ELEMENT NAME:	RECEIVED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	UP TO 128 BYTES OF DATA FROM THE CALLED HOST TO THE CALLING HOST
NOTES:	CALLING NODE PROTOCOL LAYER

DATA ELEMENT NAME:	RECEIVED-CALLED-NODE-INTERRUPT-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (00100011) INTERRUPT DATA IS ANY STRING OF UP TO 8 BITS THAT IS TRANSMITTED FROM THE CALLED HOST TO THE CALLING HOST
NOTES:	CALLING NODE PROTOCOL LAYER

DATA ELEMENT NAME:	RECEIVED-CALLING-HOST-CLEAR-REQUEST-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 1 0 0 1 1)
NOTES:	CALLING HOST AND NODE LAYERS

DATA ELEMENT NAME:	RECEIVED-CALLING-HOST-INTERRUPT-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	ANY STRING OF UP TO 8 BITS THAT IS TRANSMITTED FROM THE CALLING HOST TO THE CALLED HOST WITHOUT USING LEVEL 3 FLOW CONTROL
NOTES:	CALLING NODE PROTOCOL LAYER

DATA ELEMENT NAME:	RECEIVED-CALLING-HOST-RESET-REQUEST
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 1 1 0 1 1)
NOTES:	CALLING HOST AND NODE PROTOCOL LAYERS

DATA ELEMENT NAME:	RECEIVED-CALLING-HOST-RESTART-REQUEST-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (1 1 1 1 1 0 1 1)
NOTES:	CALLING HOST AND NODE PROTOCOL LAYERS

DATA ELEMENT NAME:	RECEIVED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	UP TO 128 BYTES OF DATA FROM THE CALLING HOST TO THE CALLED HOST
NOTES:	CALLED NODE PROTOCOL LAYER

DATA ELEMENT NAME:	RECEIVED-CALLING-NODE-INTERRUPT-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 1 0 0 0 1 1)
NOTES:	CALLED NODE PROTOCOL LAYER

DATA ELEMENT NAME:	RECEIVED-INCOMING-CALL-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 0 1 0 1 1)
NOTES:	CALLED NODE PROTOCOL LAYER

DATAFLOW NAME:	RECEIVED-WINDOWED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
ALIASES:	NONE
COMPOSITION:	RECEIVED-WINDOWED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET = P(R) + P(S) + CALLED-HOST-TO-CALLING-HOST-DATA
NOTES:	CALLED NODE PROTOCOL LAYER

DATAFLOW NAME:	RECEIVED-WINDOWED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
ALIASES:	NONE
COMPOSITION:	RECEIVED-WINDOWED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET = P(R) + P(S) + CALLING-HOST-TO-CALLED-HOST-DATA
NOTES:	CALLING NODE PROTOCOL LAYER

DATA ELEMENT NAME:	RELAYED-CALL-ACCEPTANCE-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 0 1 1 1 1)
NOTES:	CALLED NODE PROTOCOL LAYER

DATA ELEMENT NAME:	RELAYED-INCOMING-CALL-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	OCTET 3 = (0 0 0 0 1 0 1 1)
NOTES:	CALLED NODE AND HOST PROTOCOL LAYERS

DATA ELEMENT NAME:	RESTART-INDICATION-PACKET
ALIASES:	ERROR-RECOVERY-PACKET
VALUES AND MEANINGS:	OCTET 3 = (1 1 1 1 1 0 1 1)
NOTES:	CALLING HOST AND NODE LAYER

DATAFLOW NAME:	ROUTED-CALLED-NODE-PACKET
ALIASES:	ROUTED-CALLED-NODE-TO-CALLING-NODE-LEVEL-3-PACKET
COMPOSITION:	SEE ALIAS
NOTES:	SEE ALIAS

DATAFLOW NAME:	ROUTED-CALLED-NODE-TO-CALLING-NODE-LEVEL-3-PACKET
ALIASES:	ROUTED-CALLED-NODE-PACKET
COMPOSITION:	ROUTED-CALLED-NODE-TO-CALLING-NODE-LEVEL-3-PACKET = [RECEIVED-CALLED-NODE-INTERRUPT-PACKET | RECEIVED-CALL-ACCEPTED-PACKET | RECEIVED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET]
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	ROUTED-CALLING-NODE-PACKET
ALIASES:	ROUTED-CALLING-NODE-TO-CALLED-NODE-LEVEL-3-PACKET
COMPOSITION:	SEE ALIAS
NOTES:	SEE ALIAS

DATAFLOW NAME:	ROUTED-CALLING-NODE-TO-CALLED-NODE-LEVEL-3-PACKET
ALIASES:	ROUTED-CALLING-NODE-PACKET
COMPOSITION:	ROUTED-CALLING-NODE-TO-CALLED-NODE-LEVEL-3-PACKET = [RECEIVED-CALLING-NODE-LEVEL-3-PACKET | RECEIVED-INCOMING-CALL-PACKET | RECEIVED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET]
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-CATEGORY-1-PACKET
ALIASES:	NONE
COMPOSITION:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-CATEGORY-1-PACKET = LAST-PACKET-BIT + CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
NOTES:	CALLING HOST PROTOCOL LAYER

DATAFLOW NAME:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-CATEGORY-2-PACKET
ALIASES:	NONE
COMPOSITION:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-CATEGORY-2-PACKET = MORE-DATA-BIT + CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
NOTES:	CALLING HOST PROTOCOL LAYER

DATAFLOW NAME:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-COMPLETE-PACKET-SEQUENCE
ALIASES:	NONE
COMPOSITION:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-COMPLETE-PACKET-SEQUENCE = CALLED-HOST-TO-CALLING-HOST-BUFFERED-SEQUENCE + TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-CATEGORY-1-PACKET
NOTES:	CALLING HOST PROTOCOL LAYER

DATA ELEMENT NAME:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-DATA
ALIASES:	NONE
VALUES AND MEANINGS:	ANY STRING OF BITS THAT HAS BEEN TRANSMITTED FROM THE CALLED HOST TO THE CALLING HOST
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
ALIASES:	NONE
COMPOSITION:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET = [TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-CATEGORY-1-PACKET | TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-CATEGORY-2-PACKET]
NOTES:	CALLING HOST PROTOCOL LAYER

DATA ELEMENT NAME:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-INTERRUPT-DATA
ALIASES:	NONE
VALUES AND MEANINGS:	ANY STRING OF 8 BITS THAT HAS BEEN TRANSMITTED FROM THE CALLED HOST TO THE CALLING HOST WITHOUT USING THE LEVEL 3 FLOW CONTROL MECHANISM
NOTES:	CALLING HOST PROTOCOL LAYER

DATAFLOW NAME:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-NODE-INTERRUPT-PACKET
ALIASES:	NONE
COMPOSITION:	TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-NODE-INTERRUPT-PACKET = NODE-INTERRUPT-IDENTIFIER + TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-INTERRUPT-DATA
NOTES:	CALLING HOST PROTOCOL LAYER

DATAFLOW NAME:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-CATEGORY-1-PACKET
ALIASES:	NONE
COMPOSITION:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-CATEGORY-1-PACKET = LAST-PACKET-BIT + CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
NOTES:	CALLED HOST PROTOCOL LAYER

DATAFLOW NAME:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-CATEGORY-2-PACKET
ALIASES:	NONE
COMPOSITION:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-CATEGORY-2-PACKET = MORE-DATA-BIT + CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
NOTES:	CALLED HOST PROTOCOL LAYER

DATAFLOW NAME:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-COMPLETE-PACKET-SEQUENCE
ALIASES:	NONE
COMPOSITION:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-COMPLETE-PACKET-SEQUENCE = CALLING-HOST-TO-CALLED-HOST-BUFFERED-SEQUENCE + TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-CATEGORY-1-PACKET
NOTES:	CALLED HOST PROTOCOL LAYER

DATA ELEMENT NAME:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-DATA
ALIASES:	NONE
VALUES AND MEANINGS:	ANY STRING OF BITS THAT HAS BEEN TRANSMITTED FROM THE CALLING HOST TO THE CALLED HOST
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
ALIASES:	NONE
COMPOSITION:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET = [TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-CATEGORY-1-PACKET | TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-CATEGORY-2-PACKET]
NOTES:	CALLED HOST PROTOCOL LAYER

DATA ELEMENT NAME:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-INTERRUPT-DATA
ALIASES:	NONE
VALUES AND MEANINGS:	ANY STRING OF 8 BITS THAT HAS BEEN TRANSMITTED FROM THE CALLING HOST TO THE CALLED HOST WITHOUT USING THE LEVEL 3 FLOW CONTROL
NOTES:	CALLED HOST PROTOCOL LAYER

DATAFLOW NAME:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-NODE-INTERRUPT-PACKET
ALIASES:	NONE
COMPOSITION:	TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-NODE-INTERRUPT-PACKET = NODE-INTERRUPT-IDENTIFIER + TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-INTERRUPT-DATA
NOTES:	CALLED HOST PROTOCOL LAYER

DATAFLOW NAME:	TRANSMITTED-DATA
ALIASES:	NONE
COMPOSITION:	TRANSMITTED-DATA= [TRANSMITTED-CALLING-HOST-TO-CALLED-HOST-DATA | TRANSMITTED-CALLED-HOST-TO-CALLING-HOST-DATA]
NOTES:	CONTEXT LAYER

DATAFLOW NAME:	WINDOWED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
ALIASES:	NONE
COMPOSITION:	WINDOWED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET = [CALLED-HOST-TO-CALLING-HOST-DATA-PACKET + PACKET-SEND-NUMBER | HOST-RR | HOST-RNR | HOST-REJ] + PACKET-RECEIVE-NUMBER
NOTES:	CALLED HOST PROTOCOL LAYER

DATAFLOW NAME:	WINDOWED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
ALIASES:	NONE
COMPOSITION:	WINDOWED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET = [CALLING-HOST-TO-CALLED-HOST-DATA-PACKET + PACKET-SEND-NUMBER | HOST-RR | HOST-RNR | HOST-REJ] + PACKET-RECEIVE-NUMBER
NOTES:	CALLING HOST PROTOCOL LAYER

FILE DEFINITIONS

FILE OR DATABASE NAME:	CALLED-HOST-P(R)
ALIASES:	NONE
COMPOSITION:	CALLED-HOST-P(R) = ARRAY OF
	[0 | 1 | 2 | 3 | 4 | 5 | 6 | 7]
	WITH 1 VALUE FOR EACH LOGICAL CHANNEL
ORGANIZATION:	ARRAY
NOTES:	CALLED HOST PROTOCOL LAYER

FILE OR DATABASE NAME:	CALLED-HOST-P(S)
ALIASES:	NONE
COMPOSITION:	CALLED-HOST-P(S) = ARRAY OF
	[0 | 1 | 2 | 3 | 4 | 5 | 6 | 7]
	WITH 1 VALUE FOR EACH LOGICAL CHANNEL
ORGANIZATION:	ARRAY
NOTES:	CALLED HOST PROTOCOL LAYER

FILE OR DATABASE NAME:	CALLED-NODE-P(R)
ALIASES:	NONE
COMPOSITION:	CALLED-NODE-P(R) = ARRAY OF
	[0 | 1 | 2 | 3 | 4 | 5 | 6 | 7]
	WITH 1 VALUE FOR EACH LOGICAL CHANNEL
ORGANIZATION:	ARRAY
NOTES:	CALLED NODE PROTOCOL LAYER

FILE OR DATABASE NAME:	CALLED-NODE-P(S)
ALIASES:	NONE
COMPOSITION:	CALLED-NODE-P(S) = ARRAY OF
	[0 | 1 | 2 | 3 | 4 | 5 | 6 | 7]
	WITH 1 VALUE FOR EACH LOGICAL CHANNEL
ORGANIZATION:	ARRAY
NOTES:	CALLED NODE PROTOCOL LAYER

FILE OR DATABASE NAME:	CALLING-HOST-P(R)
ALIASES:	NONE
COMPOSITION:	CALLING-HOST-P(R) = ARRAY OF
	[0 | 1 | 2 | 3 | 4 | 5 | 6 | 7]
	WITH 1 VALUE FOR EACH LOGICAL CHANNEL
ORGANIZATION:	ARRAY
NOTES:	CALLING HOST PROTOCOL LAYER

```
FILE OR DATABASE NAME: CALLING-HOST-P(S)
ALIASES:               NONE
COMPOSITION:           CALLING-HOST-P(S) = ARRAY OF
                        [0 | 1 | 2 | 3 | 4 | 5 | 6 | 7]
                        WITH 1 VALUE FOR EACH LOGICAL CHANNEL
ORGANIZATION:          ARRAY
NOTES:                 CALLING HOST PROTOCOL LAYER


FILE OR DATABASE NAME: CALLING-NODE-P(R)
ALIASES:               NONE
COMPOSITION:           CALLING-NODE-P(R) = ARRAY OF
                        [0 | 1 | 2 | 3 | 4 | 5 | 6 | 7]
                        WITH 1 VALUE FOR EACH LOGICAL CHANNEL
ORGANIZATION:         ARRAY
NOTES:                 CALLING NODE PROTOCOL LAYER


FILE OR DATABASE NAME: CALLING-NODE-P(S)
ALIASES:               NONE
COMPOSITION:           CALLING-NODE-P(S) = ARRAY OF
                        [0 | 1 | 2 | 3 | 4 | 5 | 6 | 7]
                        WITH 1 VALUE FOR EACH LOGICAL CHANNEL
ORGANIZATION:         ARRAY
NOTES:                 CALLING NODE PROTOCOL LAYER
```

PROCESS SPECIFICATIONS

PROCESS NAME: DIVIDE DATA INTO PACKETS
PROCESS NUMBER: 1.1
PROCESS DESCRIPTION:

```
For each 128 bytes
  Make DATA-PACKET
  If more bytes remain then use MORE-DATA-BIT
  Else use LAST-PACKET-BIT
```

PROCESS NAME: PLACE CHANNNEL IN DATA TRANSFER STATE
PROCESS NUMBER: 1.2
PROCESS DESCRIPTION:
Execute NODE-CALL-SETUP-PACKET by making state transition specified by the table below:

Present State	Next State			
	Call-Cnc	Clr-I	Reset-I	Restart-I
Rdy	Error	Error	Error	P-Restart
Dat-Tr				
a) P-Reset	Error	P-Clr	FC-Rdy	P-Restart
b) FC-Rdy	Error	P-Clr	P-Reset	P-Restart
P-Clr	P-Clr	Rdy	P-Clr	P-Restart
P-Restart	P-Restart	P-Restart	P-Restart	Rdy
Call-Set	FC-Rdy	Rdy	Error	P-Restart

Present State	Next State		
	Reset-C	Restart-C	Clr-C
Rdy	Error	Error	Error
Dat-Tr			
a) P-Reset	FC-Rdy	Error	Error
b) FC-Rdy	Error	Error	Error
P-Clr	Error	Error	Rdy
P-Restart	Error	Rdy	Error
Call-Set	Error	Error	Error

Where

Call-Cnc	= Call-Connected-Packet
Clr-I	= Clear-Indication-Packet
Reset-I	= Reset-Indication-Packet
Restart-I	= Restart-Indication-Packet
Reset-C	= Reset-Confirmation-Packet
Clr-C	= Clear-Confirmation-Packet

```
  Rdy = Ready State
  Dat-Tr = Data Transfer State
  P-Reset = Pending Reset State
  FC-Rdy = Flow Control Ready State
  P-Clr = Pending Clear State
  P-Restart = Pending Restart State
  Call-Set = Call Setup State

Case state of

  Ready:  If CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
           available then
            Set up call by sending CALL-REQUEST-PACKET and
             entering Call Setup State
  Pending
   Reset: If RESET-REQUEST-PACKET has not been sent or if
           max-time-for-response has elapsed then
           send RESET-REQUEST-PACKET and restart timer
  Flow
  Control
   Ready: Queue CALLING-HOST-TO-CALLED-HOST-DATA-PACKET for
           channel
  Pending
   Clear: If CLEAR-REQUEST-PACKET has not been sent or if
           max-time-for-response has elapsed then
           send CLEAR-REQUEST-PACKET and restart timer
  Pending
 Restart: If RESTART-REQUEST-PACKET has not been sent or if
           max-time-for-response has elapsed then
           send RESTART-REQUEST-PACKET and restart timer
  Call
  Setup: If max-time-for-response has elapsed then send
          CALL-REQUEST-PACKET and restart timer

 Error:  Send RESTART-REQUEST-PACKET and enter Pending
          Restart State


PROCESS NAME:          WINDOW CALLING-HOST-TO-CALLED-HOST-
                        DATA-PACKET
PROCESS NUMBER:        1.3
PROCESS DESCRIPTION:
Case of CALLING-NODE-SUPERVISORY-PACKET
  NODE-RR-PACKET:  Calling Node State = Ready
                   P(R) = P(R) of NODE-RR-PACKET
  NODE-RNR-PACKET: Calling Node State = Not Ready
                   P(R) = P(R) of NODE-RNR-PACKET
  NODE-REJ-PACKET: Calling Node State = Ready
                   P(R) = P(R) of NODE-REJ-PACKET
While P(S)<P(R) then Send a CALLING-HOST-TO-CALLED-HOST-
  DATA-PACKET from the queue and increment P(S) modulo 8
```

PROCESS NAME: SEND PACKET TO CALLING NODE
PROCESS NUMBER: 1.4
PROCESS DESCRIPTION:
Place Packet in Shared Memory Priority Queue for CALLING NODE
Priority 1: CALLING-HOST-INTERRUPT-CONFIRMATION-PACKET
Priority 2: HOST-CALL-SETUP-PACKET
Priority 3: WINDOWED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
(Priority 1 is highest)

PROCESS NAME: DETERMINE CALLING-NODE-TO-CALLING-HOST-PACKET TYPE
PROCESS NUMBER: 1.5
PROCESS DESCRIPTION:
If Octet 3 = (00001011) or (00001111) or (00010011) or (00010111) then CALLING-NODE-TO-CALLING-HOST-PACKET is a NODE-CALL-SETUP-PACKET
Else if Octet 3 = (XXX00001) or (XXX00101) or (XXX01001) or (00011011) or (00011111) then CALLING-NODE-TO-CALLING-HOST-PACKET is a CALLING-NODE-SUPERVISORY-PACKET
Else if Octet 3 = (XXXXXXX0) then CALLING-NODE-TO-CALLING-HOST-PACKET is a CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
Else if Octet 3 = (00100011) then CALLING-NODE-TO-CALLING-HOST-PACKET is a CALLED-HOST-TO-CALLING-HOST-NODE-INTERRUPT-PACKET

PROCESS NAME: DETERMINE DATA PACKET CATEGORY AND EXTRACT FLOW CONTROL
PROCESS NUMBER: 1.6
PROCESS DESCRIPTION:
P(R) = Bits 8,7,6 of Octet 3 of CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
P(S) = Bits 4,3,2 of Octet 3 of CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
If Bit 5 of Octet 3 of CALLED-HOST-TO-CALLING-HOST-DATA-PACKET = 0 then CALLED-HOST-TO-CALLING-HOST-DATA-PACKET is a CATEGORY-1-PACKET
Else CALLED-HOST-TO-CALLING-HOST-DATA-PACKET is a CATEGORY-2-PACKET

PROCESS NAME: BUFFER PACKET SEQUENCE
PROCESS NUMBER: 1.7
PROCESS DESCRIPTION:
Extract CALLED-HOST-TO-CALLING-HOST Data Field and append to end of Data Fields already buffered

```
PROCESS NAME:          ASSEMBLE PACKET SEQUENCE
PROCESS NUMBER:        1.8
PROCESS DESCRIPTION:
When a CATEGORY-1-PACKET is received
 Append Data Field to end of BUFFERED-SEQUENCE


PROCESS NAME:          CONFIRM RECEIPT OF INTERRUPT PACKET
PROCESS NUMBER:        1.9
PROCESS DESCRIPTION:
Send HOST-INTERRUPT-CONFIRMATION-PACKET
Extract INTERRUPT-USER-DATA from CALLED-HOST-TO-CALLING-
 HOST-NODE-INTERRUPT-PACKET which is contained in Octet 4


PROCESS NAME:          SEND DATA TO CALLING HOST
PROCESS NUMBER:        1.10
PROCESS DESCRIPTION:
Place CALLED-HOST-TO-CALLING-HOST-INTERRUPT-DATA and
 CALLED-HOST-TO-CALLING-HOST-COMPLETE-PACKET-SEQUENCE in
 priority queue for transmission to CALLING HOST by
 VIRTUAL TERMINAL PROTOCOL
Priority 1 (highest)  INTERRUPT-DATA
Priority 2            COMPLETE-PACKET-SEQUENCE


PROCESS NAME:          DETERMINE CALLING-HOST-PACKET TYPE
PROCESS NUMBER:        2.1.1
PROCESS DESCRIPTION:
Case Octet 3 of
(00001011):    RECEIVED-CALL-REQUEST-PACKET
(XXXXXXX0):    RECEIVED-WINDOWED-CALLING-HOST-TO-CALLED-HOST-
                DATA-PACKET
(00100011):    RECEIVED-CALLING-HOST-INTERRUPT-PACKET
(00010011):    RECEIVED-CALLING-HOST-CLEAR-REQUEST
(00011011):    RECEIVED-CALLING-HOST-RESET-REQUEST
(11111011):    RECEIVED-CALLING-HOST-RESTART-REQUEST
(00010111) or
(00100111) or
(00011111) or
(11111111):    CALLING-HOST-CONFIRMATION-PACKET
(XXX00001) or
(XXX00101) or
(XXX01001):    CALLING-HOST-SUPERVISORY-PACKET
otherwise:     INVALID-PACKET-TYPE
```

```
PROCESS NAME:          NOTIFY CALLED NODE OF INCOMING CALL
PROCESS NUMBER:        2.1.2
PROCESS DESCRIPTION:
Set State of CALLING NODE to Call Setup State
Send CALL-REQUEST-PACKET as INCOMING-CALL-PACKET to
 CALLED NODE


PROCESS NAME:          UPDATE CALLING NODE WINDOWING VARIABLES
PROCESS NUMBER:        2.1.3
PROCESS DESCRIPTION:
P(R) = Bits 8,7,6 of Octet 3 of RECEIVED-WINDOWED-CALLING-
        HOST-TO-CALLED-HOST-DATA-PACKET
P(S) = Bits 4,3,2 of Octet 3 of RECEIVED-WINDOWED-CALLING-
        HOST-TO-CALLED-HOST-DATA-PACKET


PROCESS NAME:          CONFIRM RECEIPT OF CALLING-HOST-
                        INTERRUPT
PROCESS NUMBER:        2.1.4
PROCESS DESCRIPTION:
Send CALLING-NODE-INTERRUPT-CONFIRMATION-PACKET to CALLING-
 HOST upon receipt of a RECEIVED-CALLING-HOST-INTERRUPT-
 PACKET


PROCESS NAME:          CONFIRM CALLING HOST CLEAR REQUEST
PROCESS NUMBER:        2.1.5
PROCESS DESCRIPTION:
Send any remaining CALLING-HOST-TO-CALLED-HOST-DATA-PACKETs
 to CALLED-NODE
Clear virtual call and set CALLING NODE state to Ready State
Send CALLING-NODE-CLEAR-CONFIRMATION-PACKET to CALLING-HOST


PROCESS NAME:          CONFIRM CALLING HOST RESET REQUEST
PROCESS NUMBER:        2.1.6
PROCESS DESCRIPTION:
Reset P(R) and P(S) to zero for the channel specified in the
  RESET-REQUEST-PACKET
Set CALLING-NODE state to Flow Control Ready state
Send CALLING-NODE-RESET-CONFIRMATION-PACKET to CALLING-HOST
```

```
PROCESS NAME:          CONFIRM CALLING HOST RESTART REQUEST
PROCESS NUMBER:        2.1.7
PROCESS DESCRIPTION:
Reset all P(S) and P(R) for the CALLING-NODE
Set the state of virtual circuits to Flow Control Ready
Set the state of virtual channels to Ready
Send CALLING-NODE-RESTART-CONFIRMATION-PACKET to CALLING-HOST


PROCESS NAME:          SEND PACKET ONTO NETWORK
PROCESS NUMBER:        2.2
PROCESS DESCRIPTION:
Place PACKET in queue for transmission by the
 Level 2 Protocol and the Routing Algorithm


PROCESS NAME:          DETERMINE CALLED-NODE-PACKET TYPE
PROCESS NUMBER:        2.3.1
PROCESS DESCRIPTION:
Case Octet 3 of
(00100011):  RECEIVED-CALLED-NODE-INTERRUPT-PACKET
(00001111):  RECEIVED-CALL-ACCEPTED-PACKET
(XXXXXXX0):  RECEIVED-CALLED-HOST-TO-CALLING-HOST-DATA-PACKET
otherwise:   INVALID-PACKET TYPE


PROCESS NAME:          INTERRUPT CALLING NODE TO CALLING HOST
                        DATA FLOW
PROCESS NUMBER:        2.3.2
PROCESS DESCRIPTION:
Transfer CALLED-HOST-INTERRUPT-DATA from RECEIVED-CALLED-
 NODE-PACKET to CALLING-NODE-INTERRUPT-PACKET


PROCESS NAME:          NOTIFY CALLING HOST OF CALL CONNECTION
PROCESS NUMBER:        2.3.3
PROCESS DESCRIPTION:
If in Call Setup state and CALL-ACCEPTED-PACKET received
 from CALLED NODE then send CALL-CONNECTED-PACKET to CALLING
 NODE
If not in Call Setup state and CALL-ACCEPTED-PACKET received
 then discard packet
If in Call Setup state and no CALL-ACCEPTED-PACKET is
 received prior to expiration of max-time-for-call-setup-
 timer then send CALLING-NODE-CLEAR-INDICATION-PACKET to
 CALLING HOST
```

```
PROCESS NAME:          WINDOW CALLED-HOST-TO-CALLING-HOST-
                        DATA-PACKETS
PROCESS NUMBER:        2.3.4
PROCESS DESCRIPTION:
Case CALLING-HOST-SUPERVISORY-PACKET of

  HOST-RR:  Set CALLING NODE state to Ready
  HOST-RNR: Set CALLING NODE state to Not Ready
  HOST-REJ: Set CALLING NODE state to Ready

Set P(R) = Bits 8,7,6 of Octet 3 of CALLING-HOST-SUPERVISORY-
  PACKET
While there are RECEIVED-CALLED-HOST-TO-CALLING-HOST-DATA-
 PACKETs available for transmission to the CALLING HOST
 and while P(S)<P(R) send CALLING-NODE-WINDOWED-CALLED-HOST-
 TO-CALLING-HOST-DATA-PACKETs to CALLING HOST
If no DATA-PACKETS available for transmission for max-time-
 between-updates then send CALLING-NODE-SUPERVISORY-PACKET
 to CALLING HOST
```

```
PROCESS NAME:          SEND PACKET TO CALLING HOST
PROCESS NUMBER:        2.4
PROCESS DESCRIPTION:
Place Packet in Shared Memory Priority Queue for CALLING-HOST
    Priority 1:  RESTART-INDICATION-PACKET
    Priority 2:  CALLED-HOST-TO-CALLING-HOST-NODE-INTERRUPT-
                  PACKET
    Priority 3:  CALLING-NODE-CONFIRMATION-PACKETS
    Priority 4:  CALLED-HOST-TO-CALLING-HOST-DATA-PACKETS
    Priority 5:  CALLING-NODE-SUPERVISORY-PACKETS
(Priority 1 is the highest)
```

```
PROCESS NAME:          RECOVER FROM PROCEDURE ERROR
PROCESS NUMBER:        2.5
PROCESS DESCRIPTION:
Upon detection of a LOCAL-PROCEDURE-ERROR send an
 ERROR-RECOVERY-PACKET to the CALLING HOST
```

PROCESS NAME: DETERMINE CALLED-HOST-PACKET TYPE
PROCESS NUMBER: 4.1.1
PROCESS DESCRIPTION:

```
Case Octet 3 of
(00001111):   RECEIVED-CALL-ACCEPTED-PACKET
(XXXXXXX0):   RECEIVED-WINDOWED-CALLED-HOST-TO-CALLING-HOST-
               DATA-PACKET
(00100011):   RECEIVED-CALLED-HOST-INTERRUPT-PACKET
(00010011):   RECEIVED-CALLED-HOST-CLEAR-REQUEST
(00011011):   RECEIVED-CALLED-HOST-RESET-REQUEST
(11111011):   RECEIVED-CALLED-HOST-RESTART-REQUEST
(00010111) or
(00100111) or
(00011111) or
(11111111):   CALLED-HOST-CONFIRMATION-PACKET
(XXX00001) or
(XXX00101) or
(XXX01001):   CALLED-HOST-SUPERVISORY-PACKET
otherwise:    INVALID-PACKET-TYPE
```

PROCESS NAME: RELAY CALL ACCEPTANCE TO CALLING NODE
PROCESS NUMBER: 4.1.2
PROCESS DESCRIPTION:
Set state of CALLED NODE to Flow Control Ready state
Send RELAYED-CALL-ACCEPTANCE-PACKET to CALLING NODE

PROCESS NAME: UPDATE CALLED NODE WINDOWING VARIABLES
PROCESS NUMBER: 4.1.3
PROCESS DESCRIPTION:

```
P(R) = Bits 8,7,6 of Octet 3 of RECEIVED-WINDOWED-CALLED-
        HOST-TO-CALLING-HOST-DATA-PACKET
P(S) = Bits 4,3,2 of Octet 3 of RECEIVED-WINDOWED-CALLED-
        HOST-TO-CALLING-HOST-DATA-PACKET
```

PROCESS NAME: CONFIRM RECEIPT OF CALLED-HOST-INTERRUPT
PROCESS NUMBER: 4.1.4
PROCESS DESCRIPTION:
Send CALLED-NODE-INTERRUPT-CONFIRMATION-PACKET to CALLED-HOST upon receipt of a RECEIVED-CALLED-HOST-INTERRUPT-PACKET

PROCESS NAME: CONFIRM CALLED HOST CLEAR REQUEST
PROCESS NUMBER: 4.1.5
PROCESS DESCRIPTION:
Send any remaining CALLED-HOST-TO-CALLING-HOST-DATA-PACKETs to CALLING NODE
Clear virtual call and set CALLED NODE state to Ready state
Send CALLED-NODE-CLEAR-CONFIRMATION-PACKET to CALLING-HOST

PROCESS NAME: CONFIRM CALLING HOST RESET REQUEST
PROCESS NUMBER: 4.1.6
PROCESS DESCRIPTION:
Reset P(R) and P(S) for the CALLED NODE
Set CALLED NODE state to Flow Control Ready state
Send CALLED-NODE-RESET-CONFIRMATION-PACKET to CALLED HOST

PROCESS NAME: CONFIRM CALLED HOST RESTART REQUEST
PROCESS NUMBER: 4.1.7
PROCESS DESCRIPTION:
Reset all P(R) and P(S) for the CALLED NODE
Set the state of the virtual circuits to Flow Control Ready
Set the state of the virtual channels to Ready
Send CALLED-NODE-RESTART-CONFIRMATION-PACKET to CALLED HOST

PROCESS NAME: SEND PACKET ONTO NETWORK
PROCESS NUMBER: 4.2
PROCESS DESCRIPTION: SEE PROCESS 2.2

PROCESS NAME: DETERMINE CALLING NODE PACKET TYPE
PROCESS NUMBER: 4.3.1
PROCESS DESCRIPTION:
```
Case Octet 3 of
(00100011):  RECEIVED-CALLING-NODE-INTERRUPT-PACKET
(00001011):  RECEIVED-INCOMING-CALL-PACKET
(XXXXXXX0):  RECEIVED-CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
otherwise:   INVALID-PACKET TYPE
```

PROCESS NAME: INTERRUPT CALLED NODE TO CALLED HOST DATA FLOW
PROCESS NUMBER: 4.3.2
PROCESS DESCRIPTION:
Transfer CALLING-HOST-INTERRUPT-DATA from RECEIVED-CALLING-NODE-INTERRUPT-PACKET to CALLED-NODE-INTERRUPT-PACKET

```
PROCESS NAME:          NOTIFY CALLED HOST OF INCOMING CALL
PROCESS NUMBER:        4.3.3
PROCESS DESCRIPTION:
If in Ready state then relay RECEIVED-INCOMING-CALL-PACKET
 to CALLING HOST and set CALLED NODE state to Call Setup
 state
Else discard packet


PROCESS NAME:          WINDOW CALLING-HOST-TO-CALLED-HOST-
                        DATA-PACKETS
PROCESS NUMBER:        4.3.4
PROCESS DESCRIPTION:
Case CALLED-HOST-SUPERVISORY-PACKET of

  HOST-RR,HOST-REJ:  Set CALLED NODE state to Ready
          HOST-RNR:  Set CALLED NODE state to Not Ready

Set P(R) = Bits 8,7,6 of Octet 3 of CALLED-HOST-SUPERVISORY-
 PACKET
While there are RECEIVED-CALLING-HOST-TO-CALLED-HOST-DATA-
 PACKETS available for transmission to the CALLED HOST and
 while P(S) < P(R) send CALLED-NODE-WINDOWED-CALLING-HOST-
 TO-CALLED-HOST-DATA-PACKETs to CALLED HOST
If no DATA-PACKETS available for transmission for max-time-
 between-updates then send CALLED-NODE-SUPERVISORY-PACKET
 to CALLED HOST


PROCESS NAME:          SEND PACKET TO CALLED HOST
PROCESS NUMBER:        4.4
PROCESS DESCRIPTION:
Place Packet in Shared Memory Priority Queue for CALLED-HOST
    Priority 1:  RESTART-INDICATION-PACKET
    Priority 2:  CALLING-HOST-TO-CALLED-HOST-NODE-INTERRUPT-
                  PACKET
    Priority 3:  CALLED-NODE-CONFIRMATION-PACKETS
    Priority 4:  CALLING-HOST-TO-CALLED-HOST-DATA-PACKETS
    Priority 5:  CALLED-NODE-SUPERVISORY-PACKETS
(Priority 1 is the highest)


PROCESS NAME:          RECOVER FROM PROCEDURE ERROR
PROCESS NUMBER:        4.5
PROCESS DESCRIPTION:
Upon detection of a PROCEDURE-ERROR send an
ERROR-RECOVERY-PACKET to the CALLED-HOST
```

PROCESS NAME: DIVIDE DATA INTO PACKETS
PROCESS NUMBER: 5.1
PROCESS DESCRIPTION: SEE PROCESS 1.1

PROCESS NAME: KEEP CHANNEL IN DATA TRANSFER STATE
PROCESS NUMBER: 5.2
PROCESS DESCRIPTION:
Execute NODE-CALL-MAINTENANCE-PACKET by making the state transition specified by the table below:

Present State	Next State			
	Inc-Call	Clr-I	Reset-I	Restart-I
Rdy	FC-Rdy	Error	Error	P-Restart
Dat-Tr				
a) P-Reset	P-Reset	P-Clr	FC-Rdy	P-Restart
b) FC-Rdy	FC-Rdy	P-Clr	P-Reset	P-Restart
P-Clr	P-Clr	Rdy	P-Clr	P-Restart
P-Restart	P-Restart	P-Restart	P-Restart	Rdy

Present State	Next State		
	Reset-C	Restart-C	Clr-C
Rdy	Error	Error	Error
Dat-Tr			
a) P-Reset	FC-Rdy	Error	Error
b) FC-Rdy	Error	Error	Error
P-Clr	Error	Error	Rdy
P-Restart	Error	Rdy	Error

Where
Inc-Call = Incoming-Call-Packet
Clr-I = Clear-Indication-Packet
Reset-I = Reset-Indication-Packet
Restart-I = Restart-Indication-Packet
Reset-C = Reset-Confirmation-Packet
Clr-C = Clear-Confirmation-Packet
Rdy = Ready State
Dat-Tr = Data Transfer State
P-Reset = Pending Reset State
FC-Rdy = Flow Control Ready State
P-Clr = Pending Clear State
P-Restart = Pending Restart State

```
Case state of

 Ready:     If INCOMING-CALL-PACKET received from CALLED
             NODE then send CALL-ACCEPTED-PACKET back to
             CALLED NODE
  Pending
   Reset: If RESET-REQUEST-PACKET has not been sent or if
           max-time-for-response has elapsed then
           send RESET-REQUEST-PACKET and restart timer
  Flow
  Control
   Ready: Queue CALLED-HOST-TO-CALLING-HOST-DATA-PACKET for
           channel
  Pending
   Clear: If CLEAR-REQUEST-PACKET has not been sent or if
           max-time-for-response has elapsed then
           send CLEAR-REQUEST-PACKET and restart timer
  Pending
 Restart: If RESTART-REQUEST-PACKET has not been sent or if
           max-time-for-response has elapsed then
           send RESTART-REQUEST-PACKET and restart timer
 Error:  Send RESTART-REQUEST-PACKET and enter Pending
          Restart State


PROCESS NAME:          WINDOW CALLED-HOST-TO-CALLING-HOST-
                        DATA-PACKET
PROCESS NUMBER:        5.3
PROCESS DESCRIPTION:
Case of CALLED-NODE-SUPERVISORY-PACKET
  NODE-RR-PACKET:  Called Node State = Ready
                   P(R) = P(R) of NODE-RR-PACKET
  NODE-RNR-PACKET: Called Node State = Not Ready
                   P(R) = P(R) of NODE-RNR-PACKET
  NODE-REJ-PACKET: Called Node State = Ready
                   P(R) = P(R) of NODE-REJ-PACKET
While P(S)<P(R) then Send a CALLED-HOST-TO-CALLING-HOST-
  DATA-PACKET from the queue and increment P(S) modulo 8


PROCESS NAME:          SEND PACKET TO CALLED NODE
PROCESS NUMBER:        5.4
PROCESS DESCRIPTION:
Place Packet in Shared Memory Priority Queue for CALLED NODE
    Priority 1:  CALLED-HOST-INTERRUPT-CONFIRMATION-PACKET
    Priority 2:  HOST-CALL-MAINTENANCE-PACKET
    Priority 3:  WINDOWED-CALLED-HOST-TO-CALLING-HOST-DATA-
                  PACKET
(Priority 1 is highest)
```

PROCESS NAME: DETERMINE CALLED-NODE-TO-CALLED-HOST PACKET TYPE
PROCESS NUMBER: 5.5
PROCESS DESCRIPTION:
If Octet 3 = (00001011) or (00010011) or (00010111) or (00011011) or (00011111) or (11111011) or (11111111) then CALLED-NODE-TO-CALLED-HOST-PACKET is a NODE-CALL-MAINTENANCE-PACKET
Else if Octet 3 = (XXX00001) or (XXX00101) or (XXX01001) or then CALLED-NODE-TO-CALLED-HOST-PACKET is a CALLED-NODE-SUPERVISORY-PACKET
Else if Octet 3 = (XXXXXXX0) then CALLED-NODE-TO-CALLED-HOST-PACKET is a CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
Else if Octet 3 = (00100011) then CALLED-NODE-TO-CALLED-HOST-PACKET is a CALLING-HOST-TO-CALLED-HOST-NODE-INTERRUPT-PACKET

PROCESS NAME: DETERMINE DATA PACKET CATEGORY AND EXTRACT FLOW CONTROL
PROCESS NUMBER: 5.6
PROCESS DEFINITION:
P(R) = Bits 8,7,6 of Octet 3 of CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
P(S) = Bits 4,3,2 of Octet 3 of CALLING-HOST-TO-CALLED-HOST-DATA-PACKET
If Bit 5 of Octet 3 of CALLING-HOST-TO-CALLED-HOST-DATA-PACKET = 0 then CALLING-HOST-TO-CALLED-HOST-DATA-PACKET is a CATEGORY-1-PACKET
Else CALLING-HOST-TO-CALLED-HOST-DATA-PACKET is a CATEGORY-2-PACKET

PROCESS NAME: BUFFER PACKET SEQUENCE
PROCESS NUMBER: 5.7
PROCESS DESCRIPTION:
Extract CALLING-HOST-TO-CALLED-HOST Data Field and append to end of Data Fields already buffered

PROCESS NAME: ASSEMBLE PACKET SEQUENCE
PROCESS NUMBER: 5.8
PROCESS DESCRIPTION: SEE PROCESS 1.8

PROCESS NAME: CONFIRM RECEIPT OF INTERRUPT PACKET
PROCESS NUMBER: 5.9
PROCESS DESCRIPTION:
Send HOST-INTERRUPT-CONFIRMATION-PACKET
Extract INTERRUPT-USER-DATA from CALLING-HOST-TO-CALLED-HOST-NODE-INTERRUPT-PACKET which is contained in Octet 4

PROCESS NAME: SEND DATA TO CALLED HOST
PROCESS NUMBER: 5.10
PROCESS DESCRIPTION:
Place CALLING-HOST-TO-CALLED-HOST-INTERRUPT-DATA and CALLING-HOST-TO-CALLED-HOST-COMPLETE-PACKET-SEQUENCE in priority queue for transmission to CALLED HOST by VIRTUAL TERMINAL PROTOCOL

Priority 1 (highest)	INTERRUPT-DATA
Priority 2	COMPLETE-PACKET-SEQUENCE

DATA DICTIONARY

FOR NETWORK PROTOCOL

DATA ELEMENT NAME:	IN-TRANSIT-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	ANY LEVEL 3 PACKET THAT HAS BEEN DETERMINED TO NOT BE AT ITS DESTINATION NODE
NOTES:	OVERVIEW LAYER

DATA ELEMENT NAME:	SOURCE-NODE-LEVEL-3-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	ANY LEVEL 3 PACKET THAT HAS BEEN PASSED FROM THE SOURCE HOST TO THE NODE SERVING THAT HOST AS AN ENTRY POINT TO THE NETWORK
NOTES:	CONTEXT LAYER

DATA ELEMENT NAME:	TRANSMITTED-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	ANY LEVEL 3 PACKET THAT HAS BEEN TRANSMITTED FROM ONE NODE TO ANOTHER
NOTES:	OVERVIEW LAYER

DATA ELEMENT NAME:	TRANSMITTED-SOURCE-NODE-LEVEL-3-PACKET
ALIASES:	TRANSMITTED-SOURCE-NODE-PACKET
VALUES AND MEANINGS:	ANY LEVEL 3 PACKET THAT HAS BEEN TRANSMITTED ACROSS THE NETWORK TO THE NODE THAT SERVES AS THE ENTRY POINT FOR THE HOST THAT IS THE PACKET'S DESTINATION
NOTES:	CONTEXT LAYER

DATA ELEMENT NAME:	TRANSMITTED-SOURCE-NODE-PACKET
ALIASES:	TRANSMITTED-SOURCE-NODE-LEVEL-3-PACKET
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	SEE ALIAS

FILE DEFINITIONS

FILE OR DATABASE NAME:	ROUTING-LOOKUP-TABLE
ALIASES:	NONE
COMPOSITION:	ROUTING-LOOKUP-TABLE = {DESTINATION-NODE + NEXT-CLOSER-NODE}
ORGANIZATION:	ARRAY EACH NODE MUST HAVE ITS OWN LOOKUP TABLE STORED IN IT
NOTES:	OVERVIEW LAYER

PROCESS SPECIFICATIONS

PROCESS NAME: DETERMINE THE NEXT CLOSER NODE
PROCESS NUMBER: 1
PROCESS DESCRIPTION:

```
Access the ROUTING-LOOKUP-TABLE using the LEVEL-3-PACKET's
 DEST-HOST field to find the NEXT-CLOSER-NODE
If the NEXT-CLOSER-NODE is the PRESENT-NODE then pass the
 LEVEL-3-PACKET to the LEVEL-3-NODE-PROTOCOL
Else output the LEVEL-3-PACKET as an IN-TRANSIT-PACKET with
 the NEXT-CLOSER-NODE as the NEXT-PACKET-DESTINATION
```

PROCESS NAME: TRANSMIT PACKET TO ROUTED NODE
PROCESS NUMBER: 2
PROCESS DESCRIPTION:

```
Use the X.25 Level 2 Protocol to transmit the IN-TRANSIT-
 PACKET to the NEXT-PACKET-DESTINATION as a TRANSMITTED-
 PACKET
```

DATA DICTIONARY

FOR X.25 LEVEL 2 PROTOCOL

DATA ELEMENT NAME:	ABORTED-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	ANY PACKET CONTAINING A SEQUENCE OF SEVEN CONSECUTIVF 1'S
NOTES:	EXTRACT VALID PACKET LAYER

DATA ELEMENT NAME:	ADDRESS-FIELD
ALIASES:	NONE
VALUES AND MEANINGS:	8 BIT FIELD (10000000) = PRIMARY TO SECONDARY COMMAND OR SECONDARY TO PRIMARY RESPONSE (11000000) = PRIMARY TO SECONDARY RESPONSE OR SECONDARY TO PRIMARY COMMAND
NOTES:	HDLC PRIMARY,SECONDARY PROTOCOL LAYERS

DATA ELEMENT NAME:	BUSY-INDICATION
ALIASES:	NONE
VALUES AND MEANINGS:	TWO-BIT CODE WITH VALUE = 10 INDICATES THAT THE SECONDARY NODE IS NOT READY TO RECEIVE LEVEL-2-PACKETS
NOTES:	EXECUTE S-FRAME RESPONSE LAYER, WINDOW PRIMARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	BUSY-N(R)
ALIASES:	N(R)
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE S-FRAME-RESPONSE LAYER, WINDOW PRIMARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	BUSY-PRIMARY-N(R)
ALIASES:	PRIMARY-N(R)
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE S-FRAME COMMAND LAYER, WINDOW SECONDARY INFORMATION BLOCKS

DATA ELEMENT NAME:	CMDR(FRMR)-RESPONSE
ALIASES:	NONE
COMPOSITION:	CMDR(FRMR)-RESPONSE = 1100 + FINAL-BIT + 001
NOTES:	HDLC PRIMARY, SECONDARY NODE PROTOCOL LAYERS

DATAFLOW NAME:	COMMAND-MODIFIER-BITS
ALIASES:	NONE
COMPOSITION:	COMMAND-MODIFIER-BITS = [DISC-COMMAND | TRANSMITTED-SABM-COMMAND]
NOTES:	EXECUTE U-FRAME COMMAND LAYER

DATAFLOW NAME:	CONTROL-FIELD
ALIASES:	NONE
COMPOSITION:	CONTROL-FIELD = [I-FRAME-CONTROL-FIELD | S-FRAME-CONTROL-FIELD | U-FRAME-CONTROL-FIELD]
NOTES:	HDLC PRIMARY, SECONDARY NODE PROTOCOL LAYERS

DATA ELEMENT NAME:	DISC-COMMAND
ALIASES:	NONE
VALUES AND MEANINGS:	BITS 1,2,7 = 1 BITS 3,4,6,8 = 0 COMMANDS THE SECONDARY NODE TO ENTER THE DISCONNECTED STATE
NOTES:	EXECUTE U-FRAME COMMAND LAYER

DATA ELEMENT NAME:	DISC-UA-RESPONSE
ALIASES:	NONE
VALUES AND MEANINGS:	DISCONNECT ACKNOWLEDGEMENT RESPONSE BITS 1,2,6,7 = 1 BITS 3,4,8 = 0 ACKNOWLEDGES TO THE PRIMARY NODE THAT THE SECONDARY NODE IS NOW IN THE DISCONNECTED STATE
NOTES:	EXECUTE U-FRAME COMMAND LAYER, WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	DM-RESPONSE
ALIASES:	NONE
VALUES AND MEANINGS:	BITS 1-4 = 1 BITS 6,7,8 = 0 INDICATES THAT THE SECONDARY NODE IS IN THE DISCONNECTED MODE AND CANNOT RECEIVE LEVEL-2-PACKETS EXCEPT FOR A SABM-COMMAND
NOTES:	EXECUTE U-FRAME RESPONSE, COMMAND LAYER AND WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	FCS-BLOCK
ALIASES:	NONE
VALUES AND MEANINGS:	FIELD CHECK SEQUENCE BLOCK A SIXTEEN BIT SEQUENCE GENERATED AT THE TRANSMITTING NODE THAT IS THE REMAINDER AFTER DIVISION OF THE LEVEL-2-PACKET BY THE CRC POLYNOMIAL
NOTES:	EXTRACT VALID PACKET LAYER

DATA ELEMENT NAME:	FCS-ERROR-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	A VALID-LENGTH-PACKET WHOSE FCS-BLOCK DOES NOT MATCH THE LOCALLY-GENERATED-CRC-POLYNOMIAL
NOTES:	EXTRACT VALID PACKET LAYER

DATA ELEMENT NAME:	FINAL-BIT
ALIASES:	I-FRAME-FINAL-BIT, S-FRAME-FINAL-BIT, U-FRAME-FINAL-BIT
VALUES AND MEANINGS:	BIT 5 OF THE SECONDARY-FRAME-CONTROL-FIELD 0 = NOT A RESPONSE TO A PRIMARY NODE POLL 1 = THE RESPONSE TO THE PRIMARY NODE POLL
NOTES:	HDLC PRIMARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	I-FRAME-FINAL-BIT
ALIASES:	FINAL-BIT
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE SECONDARY I-FRAME PACKET LAYER

DATAFLOW NAME:	I-FRAME-PACKET-SEQUENCE-INFO
ALIASES:	SECONDARY-I-FRAME-PACKET-SEQUENCE-INFO, PRIMARY-I-FRAME-PACKET-SEQUENCE-INFO
COMPOSITION:	I-FRAME-PACKET-SEQUENCE-INFO = N(R) + N(S)
NOTES:	HDLC PRIMARY, SECONDARY NODE PROTOCOL LAYERS

DATA ELEMENT NAME:	I-FRAME-POLL-BIT
ALIASES:	POLL-BIT
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE PRIMARY I-FRAME PACKET LAYER, WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	INVALID-LENGTH-PACKET
ALIASES:	INVALID-LENGTH-PRIMARY-PACKET, INVALID-LENGTH-SECONDARY-PACKET
VALUES AND MEANINGS:	ANY UNSTUFFED-PACKET THAT IS LESS THAN 32 BITS LONG
NOTES:	EXTRACT VALID PACKET LAYER

DATA ELEMENT NAME:	INVALID-LENGTH-PRIMARY-PACKET
ALIASES:	INVALID-LENGTH-PACKET
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	INVALID-LENGTH-SECONDARY-PACKET
ALIASES:	INVALID-LENGTH-PACKET
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	HDLC PRIMARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	INVALID-SECONDARY-N(R)-I-FRAME
ALIASES:	NONE
VALUES AND MEANINGS:	ANY SECONDARY-TO-PRIMARY-I-FRAME WITH AN N(R) THAT IS NOT IN THE WINDOW OF THE PRIMARY NODE
NOTES:	HDLC PRIMARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	INVALID-PRIMARY-PACKET-TYPE
ALIASES:	NONE
VALUES AND MEANINGS:	ANY PRIMARY-LEVEL-2-PACKET THAT PASSES THE INITIAL VALIDATION IN THE EXTRACT VALID PACKET PROCESS BUT CANNOT BE RECOGNIZED AS ANY OF THE LEGAL PRIMARY PACKET TYPES
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	INVALID-SECONDARY-PACKET-TYPE
ALIASES:	NONE
VALUES AND MEANINGS:	ANY SECONDARY-LEVEL-2-PACKET THAT PASSES THE INITIAL VALIDATION IN THE EXTRACT VALID PACKET PROCESS BUT CANNOT BE RECOGNIZED AS ANY OF THE LEGAL SECONDARY PACKET TYPES
NOTES:	HDLC PRIMARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	LEVEL-2-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	THE CONTENTS BETWEEN TWO SYN CHARACTERS EXCLUDING THE ABORTED PACKETS
NOTES:	EXTRACT VALID PACKET LAYER

DATA ELEMENT NAME:	LINK-STATUS
ALIASES:	NONE
VALUES AND MEANINGS:	INDICATES THE CURRENT STATE OF THE LINK BETWEEN THE PRIMARY AND SECONDARY NODES THE STATE OF THE LINK MAY BE EITHER DISCONNECTED, READY, OR PENDING-SETUP
NOTES:	EXECUTE U-FRAME RESPONSE LAYER, WINDOW PRIMARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	LOCALLY-GENERATED-CRC-REMAINDER
ALIASES:	NONE
VALUES AND MEANINGS:	A SIXTEEN BIT SEQUENCE THAT IS THE REMAINDER AFTER DIVISION OF THE VALID-LENGTH-PACKET BY THE CRC POLYNOMIAL
NOTES:	EXTRACT VALID PACKET LAYER

DATA ELEMENT NAME:	N(R)
ALIASES:	READY-N(R), BUSY-N(R), REJECT-N(R)
VALUES AND MEANINGS:	RECEIVE SEQUENCE NUMBER BITS 6,7,8 OF THE SECONDARY-I-FRAME-CONTROL-FIELD INDICATES THE NEXT PACKET THAT THE SECONDARY NODE EXPECTS TO RECEIVE FROM THE PRIMARY NODE
NOTES:	HDLC PRIMARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	N(S)
ALIASES:	NONE
VALUES AND MEANINGS:	SEND SEQUENCE NUMBER BITS 2,3,4 OF THE I-FRAME-CONTROL-FIELD
NOTES:	HDLC PRIMARY AND SECONDARY NODES PROTOCOL LAYERS

DATA ELEMENT NAME:	OUT-OF-SEQUENCE-PRIMARY-I-FRAME
ALIASES:	OUT-OF-SEQUENCE-PRIMARY-I-FRAME-PACKET
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE PRIMARY I-FRAME PACKET

DATA ELEMENT NAME:	OUT-OF-SEQUENCE-PRIMARY-I-FRAME-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	ANY PRIMARY-TO-SECONDARY-I-FRAME WITH AN N(R) THAT IS NOT IN THE WINDOW OF THE SECONDARY NODE
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	POLL-BIT
ALIASES:	I-FRAME-POLL-BIT, S-FRAME-POLL-BIT, U-FRAME-POLL-BIT
VALUES AND MEANINGS:	BIT 5 OF THE PRIMARY-FRAME-CONTROL-FIELD IF THIS BIT IS A 1 THEN THE PRIMARY NODE IS REQUESTING A STATUS UPDATE FROM THE SECONDARY NODE ON THE VALUE OF THE WINDOWING VARIABLES AND THE STATE OF THE SECONDARY NODE
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	PRIMARY-BUSY-INDICATION
ALIASES:	NONE
VALUES AND MEANINGS:	TWO-BIT CODE WITH VALUE = 10 INDICATES THAT THE PRIMARY NODE IS NOT READY TO RECEIVE LEVEL-2-PACKETS
NOTES:	EXECUTE S-FRAME COMMAND LAYER, WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	PRIMARY-BUSY-N(R)
ALIASES:	BUSY-PRIMARY-N(R)
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	WINDOW SECONDARY INFORMATION BLOCKS LAYER
DATAFLOW NAME:	PRIMARY-I-FRAME-CONTROL-FIELD
ALIASES:	NONE
COMPOSITION:	PRIMARY-I-FRAME-CONTROL-FIELD = 0 + N(S) + I-FRAME-POLL-BIT + N(R)
NOTES:	EXECUTE PRIMARY I-FRAME PACKET LAYER

DATAFLOW NAME:	PRIMARY-I-FRAME-PACKET
ALIASES:	PRIMARY-TO-SECONDARY-I-FRAME-PACKET
COMPOSITION:	SEE ALIAS
NOTES:	EXECUTE PRIMARY I-FRAME PACKET

DATA ELEMENT NAME:	PRIMARY-I-FRAME-N(R)
ALIASES:	PRIMARY-I-FRAME-PACKET-SEQUENCE-INFO
VALUES AND MEANINGS:	AN INTEGER BETWEEN 0 AND 7 INDICATING THE NEXT PACKET SEQUENCE NUMBER THAT THE PRIMARY NODE IS EXPECTING TO RECEIVE
NOTES:	EXECUTE PRIMARY I-FRAME PACKET LAYER, WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	PRIMARY-I-FRAME-PACKET-SEQUENCE-INFO
ALIASES:	PRIMARY-I-FRAME-N(R)
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

DATAFLOW NAME:	PRIMARY-INCOMING-BIT-STREAM
ALIASES:	NONE
COMPOSITION:	PRIMARY-INCOMING-BIT-STREAM = {{SYN} + TRANSMITTED-SECONDARY-LEVEL-2-PACKET}
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	PRIMARY-LEVEL-2-PACKET
ALIASES:	NONE
COMPOSITION:	PRIMARY-LEVEL-2-PACKET = ADDRESS-FIELD + CONTROL-FIELD + (INFO-FIELD)
NOTES:	HDLC PRIMARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	PRIMARY-LEVEL-3-PACKET
ALIASES:	PRIMARY-NODE-LEVEL-3-PACKET
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	WINDOW PRIMARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	PRIMARY-NODE-LEVEL-3-PACKET
ALIASES:	PRIMARY-LEVEL-3-PACKET
VALUES AND MEANINGS:	ANY LEVEL 3 PACKET AT A NODE THAT HAS BEEN DESIGNATED AS A PRIMARY NODE
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	PRIMARY-NODE-STATUS
ALIASES:	NONE
COMPOSITION:	PRIMARY-NODE-STATUS = [READY-INDICATION + READY-N(R) | BUSY-INDICATION + BUSY-N(R) | REJECTION-EXCEPTION-CONDITION + REJECT-N(R)] + S-FRAME-POLL-BIT
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	PRIMARY-N(R)
ALIASES:	READY-PRIMARY-N(R), BUSY-PRIMARY-N(R), REJECT-PRIMARY-N(R)
VALUES AND MEANINGS:	RECEIVE SEQUENCE NUMBER BITS 6,7,8 OF THE PRIMARY-I-FRAME-CONTROL-FIELD INDICATES THE NEXT PACKET THAT THE PRIMARY NODE EXPECTS TO RECEIVE FROM THE SECONDARY NODE
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

DATAFLOW NAME:	PRIMARY-OUTGOING-BIT-STREAM
ALIASES:	NONE
COMPOSITION:	PRIMARY-OUTGOING-BIT-STREAM = {{SYN} + TRANSMITTED-SECONDARY-LEVEL-2-PACKET}
NOTES:	OVERVIEW LAYER

DATA ELEMENT NAME:	PRIMARY-READY-INDICATION
ALIASES:	NONE
VALUES AND MEANINGS:	TWO BIT CODE WITH VALUE = 00 INDICATES THAT THE PRIMARY NODE IS READY TO RECEIVE LEVEL-2-PACKETS
NOTES:	EXECUTE S-FRAME COMMAND LAYER, WINDOW SECONDARY INFORMATION BLOCKS

DATA ELEMENT NAME:	PRIMARY-READY-N(R)
ALIASES:	READY-PRIMARY-N(R)
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	PRIMARY-REJECT-N(R)
ALIASES:	PRIMARY-N(R)
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE S-FRAME COMMAND LAYER, WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	PRIMARY-REJECTION-CONDITION
ALIASES:	NONE
VALUES AND MEANINGS:	TWO-BIT CODE WITH VALUE = 01 THAT INDICATES THAT THE SECONDARY-V(S) IS TO BE RESET TO THE N(R) IN THE REJ-COMMAND AND ALL PACKETS SUBSEQUENT TO THAT PACKET NUMBER RETRANSMITTED
NOTES:	EXECUTE S-FRAME COMMAND LAYER, WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATAFLOW NAME:	PRIMARY-TO-SECONDARY-I-FRAME-PACKET
ALIASES:	PRIMARY-I-FRAME-PACKET
COMPOSITION:	PRIMARY-TO-SECONDARY-I-FRAME = [INVALID-PRIMARY-I-FRAME | VALID-PRIMARY-I-FRAME-PACKET]
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	READY-INDICATION
ALIASES:	NONE
VALUES AND MEANINGS:	TWO BIT CODE WITH VALUE = 00 INDICATES THAT THE SECONDARY NODE IS READY TO RECEIVE LEVEL-2-PACKETS
NOTES:	EXECUTE S-FRAME RESPONSE LAYER, WINDOW PRIMARY INFORMATION BLOCKS

DATA ELEMENT NAME:	READY-N(R)
ALIASES:	N(R)
VALUES AND MEANINGS:	SEE ALIAS THE PACKET THAT THE SECONDARY NODE IS EXPECTING TO RECEIVE NEXT
NOTES:	EXECUTE S-FRAME RESPONSE LAYER, WINDOW PRIMARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	READY-PRIMARY-N(R)
ALIASES:	PRIMARY-N(R)
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE S-FRAME COMMAND LAYER, WINDOW SECONDARY INFORMATION BLOCKS

DATA ELEMENT NAME:	REJ-READY-INDICATION
ALIASES:	NONE
VALUES AND MEANINGS:	THIS IS A SIGNAL THAT THE REJECTION EXCEPTION CONDITION HAS BEEN CLEARED
NOTES:	WINDOW PRIMARY, SECONDARY INFORMATION BLOCKS LAYERS

DATAFLOW NAME:	REJ-COMMAND
ALIASES:	NONE
COMPOSITION:	REJ-COMMAND = PRIMARY-REJECTION-CONDITION + REJECT-PRIMARY-N(R)
NOTES:	EXECUTE S-FRAME COMMAND LAYER

DATAFLOW NAME:	REJ-RESPONSE
ALIASES:	NONE
COMPOSITION:	REJ-RESPONSE = REJECTION-EXCEPTION-CONDITION + REJECT-N(R)
NOTES:	EXECUTE S-FRAME RESPONSE LAYER

DATA ELEMENT NAME:	REJECT-N(R)
ALIASES:	N(R)
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE S-FRAME RESPONSE LAYER, WINDOW PRIMARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	REJECTION-EXCEPTION-CONDITION
ALIASES:	NONE
VALUES AND MEANINGS:	TWO-BIT CODE WITH VALUE = 01 THAT INDICATES THAT THE PRIMARY-V(S) IS TO BE RESET TO THE N(R) IN THE REJ-RESPONSE AND ALL PACKETS SUBSEQUENT TO THAT PACKET NUMBER RETRANSMITTED
NOTES:	EXECUTE S-FRAME RESPONSE LAYER, WINDOW PRIMARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	RESET-COMMAND-PACKET
ALAISES:	SABM-COMMAND
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	WINDOW PRIMARY INFORMATION BLOCKS LAYER

DATAFLOW NAME:	RESPONSE-MODIFIER-BITS
ALIASES:	NONE
COMPOSITION:	RESPONSE-MODIFIER-BITS = [DM-RESPONSE | TRANSMITTED-SECONDARY-CMDR(FRMR)-RESPONSE | UA-RESPONSE]
NOTES:	EXECUTE U-FRAME RESPONSE LAYER

DATAFLOW NAME:	RESPONSE-SUPERVISORY-FUNCTION-BITS
ALIASES:	NONE
COMPOSITION:	RESPONSE-SUPERVISORY-FUNCTION-BITS = [RR-RESPONSE | RNR-RESPONSE | REJ-RESPONSE]
NOTES:	EXECUTE S-FRAME RESPONSE LAYER

DATAFLOW NAME:	RNR-COMMAND
ALIASES:	NONE
COMPOSITION:	RNR-COMMAND = PRIMARY-BUSY-INDICATION + BUSY-PRIMARY-N(R)
NOTES:	EXECUTE S-FRAME COMMAND LAYER

DATAFLOW NAME:	RNR-RESPONSE
ALIASES:	NONE
COMPOSITION:	RNR-RESPONSE = BUSY-INDICATION + N(R)
NOTES:	EXECUTE S-FRAME RESPONSE LAYER, WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATAFLOW NAME:	RR-COMMAND
ALIASES:	NONE
COMPOSITION:	RR-COMMAND = PRIMARY-READY-INDICATION + READY-PRIMARY-N(R)
NOTES:	WINDOW PRIMARY INFORMATION BLOCKS, EXECUTE S-FRAME COMMAND LAYER LAYER

DATAFLOW NAME:	RR-RESPONSE
ALIASES:	NONE
COMPOSITION:	RR-RESPONSE = READY-INDICATION + N(R)
NOTES:	EXECUTE S-FRAME RESPONSE LAYER, WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATAFLOW NAME:	S-FRAME-COMMAND
ALIASES:	NONE
COMPOSITION:	S-FRAME-COMMAND = S-FRAME-POLL-BIT + S-FRAME-CONTROL-FIELD
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

DATAFLOW NAME:	S-FRAME-CONTROL-FIELD
ALIASES:	NONE
COMPOSITION:	S-FRAME-CONTROL-FIELD = [RR-COMMAND | RNR-COMMAND | REJ-COMMAND]
NOTES:	EXECUTE S-FRAME COMMAND LAYER

DATA ELEMENT NAME:	S-FRAME-FINAL-BIT
ALIASES:	FINAL-BIT
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE S-FRAME RESPONSE LAYER

DATA ELEMENT NAME:	S-FRAME-POLL-BIT
ALIASES:	POLL-BIT
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE S-FRAME COMMAND LAYER, WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATAFLOW NAME:	S-FRAME-RESPONSE
ALIASES:	SECONDARY-TO-PRIMARY-S-FRAME-RESPONSE
COMPOSITION:	SEE ALIAS
NOTES:	EXECUTE S-FRAME RESPONSE LAYER

DATA ELEMENT NAME:	SABM-COMMAND
ALIASES:	RESET-COMMAND-PACKET
VALUES AND MEANINGS:	SET ASYNCHRONOUS BALANCED MODE COMMAND
	CONTROL-FIELD = 1111 + POLL-BIT + 100
	RESETS THE LINK TO ENABLE RECOVERY FROM PROCEDURE ERRORS OR INITIALIZATION OF THE LINK AT STARTUP
NOTES:	HDLC PRIMARY, SECONDARY NODE PROTOCOL LAYERS

DATA ELEMENT NAME:	SABM-UA-RESPONSE
ALIASES:	NONE
VALUES AND MEANINGS:	SABM ACKNOWLEDGEMENT RESPONSE
	BITS 1,2,6,7 = 1
	BITS 3,4,8 = 0
	CONFIRMS TO THE PRIMARY NODE THAT THE SECONDARY NODE HAS NOW RESET ITS WINDOWING VARIABLES AND IS READY TO RESTART TRANSMISSION OF THE LEVEL-2-PACKETS
NOTES:	EXECUTE U-FRAME COMMAND LAYER, WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	SECONDARY-FINAL-BIT
ALIASES:	NONE
VALUES AND MEANINGS:	0 = THE SECONDARY LEVEL 2 PACKET IS NOT A RESPONSE TO A POLL FROM THE PRIMARY NODE
	1 = THE SECONDARY LEVEL 2 PACKET IS A RESPONSE TO A POLL FROM THE PRIMARY NODE
NOTES:	WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATAFLOW NAME:	SECONDARY-I-FRAME-CONTROL-FIELD
ALIASES:	NONE
COMPOSITION:	SECONDARY-I-FRAME-CONTROL-FIELD = 0 + N(S) + I-FRAME-FINAL-BIT + N(R)
NOTES:	EXECUTE SECONDARY I-FRAME PACKET LAYER

DATAFLOW NAME:	SECONDARY-I-FRAME-PACKET
ALIASES:	SECONDARY-TO-PRIMARY-I-FRAME-PACKET
COMPOSITION:	SEE ALIAS
NOTES:	EXECUTE SECONDARY I-FRAME PACKET LAYER

DATAFLOW NAME:	SECONDARY-I-FRAME-PACKET-SEQUENCE-INFO
ALIASES:	I-FRAME-PACKET-SEQUENCE-INFO
COMPOSITION:	SEE ALIAS
NOTES:	EXECUTE SECONDARY I-FRAME PACKET LAYER

DATAFLOW NAME:	SECONDARY-INCOMING-BIT-STREAM
ALIASES:	NONE
COMPOSITION:	SECONDARY-INCOMING-BIT-STREAM = {{SYN} + TRANSMITTED-PRIMARY-LEVEL-2-PACKET}
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	SECONDARY-LEVEL-2-PACKET
ALIASES:	NONE
COMPOSITION:	SECONDARY-LEVEL-2-PACKET = ADDRESS-FIELD + CONTROL-FIELD + (INFO-FIELD)
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	SECONDARY-LEVEL-3-PACKET
ALIASES:	SECONDARY-NODE-LEVEL-3-PACKET
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATA ELEMENT NAME:	SECONDARY-NODE-LEVEL-3-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	ANY LEVEL 3 PACKET THAT IS TO BE TRANSMITTED FROM A SECONDARY NODE TO A PRIMARY NODE
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	SECONDARY-NODE-STATUS
ALIASES:	NONE
COMPOSITION:	SECONDARY-NODE-STATUS = [READY-INDICATION + READY-N(R) | BUSY-INDICATION + BUSY-N(R) | REJECTION-EXCEPTION-CONDITION + REJECT-N(R)] + S-FRAME-FINAL-BIT
NOTES:	HDLC PRIMARY NODE PROTOCOL LAYER

DATAFLOW NAME:	SECONDARY-OUTGOING-BIT-STREAM
ALIASES:	NONE
COMPOSITION:	SECONDARY-OUTGOING-BIT-STREAM = {{SYN} + SECONDARY-LEVEL-2-PACKET}
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	SECONDARY-TO-PRIMARY-I-FRAME
ALIASES:	SECONDARY-I-FRAME-PACKET
COMPOSITION:	SECONDARY-TO-PRIMARY-I-FRAME = [INVALID-SECONDARY-I-FRAME | VALID-SECONDARY-I-FRAME-PACKET]
NOTES:	HDLC PRIMARY NODE PROTOCOL LAYER

DATAFLOW NAME:	SECONDARY-TO-PRIMARY-S-FRAME-RESPONSE
ALIASES:	S-FRAME-RESPONSE
COMPOSITION:	SECONDARY-TO-PRIMARY-S-FRAME-RESPONSE= S-FRAME-FINAL-BIT + RESPONSE-SUPERVISORY-FUNCTION-BITS
NOTES:	HDLC PRIMARY NODE PROTOCOL LAYER

DATAFLOW NAME:	SECONDARY-TO-PRIMARY-U-FRAME-RESPONSE
ALIASES:	U-FRAME-RESPONSE
COMPOSITION:	SECONDARY-TO-PRIMARY-U-FRAME-RESPONSE= RESPONSE-MODIFIER-BITS + U-FRAME-FINAL-BIT
NOTES:	HDLC PRIMARY NODE PROTOCOL LAYER

DATALOW NAME:	SECONDARY-VALID-PACKET
ALIASES:	NONE
COMPOSITION:	SECONDARY-VALID-PACKET = ADDRESS-FIELD + CONTROL-FIELD + (INFO-FIELD)
NOTES:	HDLC PRIMARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	SYN
ALIASES:	NONE
VALUES AND MEANINGS:	SYNCHRONIZATION CHARACTER THE OCTET (01111110)
NOTES:	OVERVIEW LAYER

DATA ELEMENT NAME:	TRANSMITTED-PRIMARY-NODE-LEVEL-3-PACKET
ALIASES:	NONE
VALUES AND MEANINGS:	ANY LEVEL 3 PACKET THAT HAS BEEN TRANSMITTED FROM A PRIMARY NODE TO A SECONDARY NODE
NOTES:	OVERVIEW LAYER

DATA ELEMENT NAME:	TRANSMITTED-SABM-COMMAND
ALIASES:	NONE
VALUES AND MEANINGS:	TRANSMITTED SET ANSYNCHRONOUS BALANCED MODE COMMAND BITS 1,2,3,4,6 = 1 BITS 7,8 = 0 COMMANDS THE SECONDARY NODE TO RESET THE LINK WINDOWING VARIABLES
NOTES:	EXECUTE U-FRAME COMMAND LAYER

DATA ELEMENT NAME:	TRANSMITTED-SECONDARY-CMDR(FRMR)-RESPONSE
ALIASES:	NONE
VALUES AND MEANINGS:	BITS 1,2,3,8 = 1 BITS 4,6,7 = 0 INDICATES THAT A LOCAL PROCEDURE ERROR HAS OCCURRED AT THE SECONDARY NODE AND REQUESTS THE PRIMARY NODE TO RESET THE LINK
NOTES:	EXECUTE U-FRAME RESPONSE LAYER

DATA ELEMENT NAME:	TRANSMITTED-SECONDARY-LEVEL-3-PACKET
ALIASES:	TRANSMITTED-SECONDARY-NODE-LEVEL-3-PACKET
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE SECONDARY I-FRAME PACKET LAYER

DATA ELEMENT NAME:	TRANSMITTED-SECONDARY-NODE-LEVEL-3-PACKET
ALIASES:	TRANSMITTED-SECONDARY-LEVEL-3-PACKET
VALUES AND MEANINGS:	ANY LEVEL 3 PACKET THAT HAS BEEN TRANSMITTED FROM A PRIMARY NODE TO A SECONDARY NODE
NOTES:	OVERVIEW LAYER

DATAFLOW NAME:	U-FRAME-COMMAND
ALIASES:	NONE
COMPOSITION:	U-FRAME-COMMAND = COMMAND-MODIFIER-BITS + U-FRAME-POLL-BIT
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

DATA ELEMENT NAME:	U-FRAME-FINAL-BIT
ALIASES:	FINAL-BIT
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE U-FRAME RESPONSE LAYER

DATA ELEMENT NAME:	U-FRAME-POLL-BIT
ALIASES:	POLL-BIT
VALUES AND MEANINGS:	SEE ALIAS
NOTES:	EXECUTE U-FRAME COMMAND LAYER, WINDOW SECONDARY INFORMATION BLOCKS LAYER

DATAFLOW NAME:	U-FRAME-RESET-COMMAND
ALIASES:	NONE
COMPOSITION:	U-FRAME-RESET-COMMAND = [SABM-COMMAND | LINK-STATUS] + U-FRAME-FINAL-BIT
NOTES:	HDLC PRIMARY NODE PROTOCOL LAYER

DATAFLOW NAME:	U-FRAME-RESPONSE
ALIASES:	SECONDARY-TO-PRIMARY-U-FRAME-RESPONSE
COMPOSITION:	SEE ALIAS
NOTES:	EXECUTE U-FRAME RESPONSE LAYER

DATA ELEMENT NAME:	UA-RESPONSE
ALIASES:	NONE
VALUES AND MEANINGS:	BITS 1,2,6,7 = 1
	BITS 3,4,8 = 0
	INDICATES THAT THE COMMAND SENT BY THE PRIMARY NODE HAS BEEN EXECUTED
NOTES:	EXECUTE U-FRAME RESPONSE LAYER

DATAFLOW NAME:	UNSTUFFED-PACKET
ALIASES:	NONE
COMPOSITION:	UNSTUFFED-PACKET = [VALID-LENGTH-PACKET | INVALID-LENGTH-PACKET]
NOTES:	EXTRACT VALID PACKET LAYER

DATA ELEMENT NAME:	UPDATED-LOWER-WINDOW-EDGE
ALIASES:	NONE
VALUES AND MEANINGS:	THIS IS AN INTEGER BETWEEN 0 AND 7 THAT IS ONE GREATER THAN THE HIGHEST PACKET SEQUENCE NUMBER THAT CAN BE SENT UNTIL THE LOWER-WINDOW-EDGE IS AGAIN UPDATED
NOTES:	WINDOW PRIMARY, SECONDARY INFORMATION BLOCKS LAYERS

DATAFLOW NAME:	VALID-LENGTH-PACKET
ALIASES:	NONE
COMPOSITION:	VALID-LENGTH-PACKET = FCS-BLOCK + [FCS-ERROR-PACKET | VALID-PACKET]
NOTES:	EXTRACT VALID PACKET LAYER

DATAFLOW NAME:	VALID-PACKET
ALIASES:	VALID-PRIMARY-PACKET, SECONDARY-VALID-PACKET
COMPOSITION:	SEE ALIASES
NOTES:	EXTRACT VALID PACKET LAYER

DATAFLOW NAME:	VALID-PRIMARY-I-FRAME-PACKET
ALIASES:	NONE
COMPOSITION:	VALID-PRIMARY-I-FRAME-PACKET = TRANSMITTED-PRIMARY-NODE-LEVEL-3-PACKET + PRIMARY-I-FRAME-CONTROL-FIELD
NOTES:	EXECUTE PRIMARY I-FRAME PACKET LAYER

DATALOW NAME:	VALID-PRIMARY-PACKET
ALIASES:	NONE
COMPOSITION:	VALID-PRIMARY-PACKET = ADDRESS-FIELD + CONTROL-FIELD + (INFO-FIELD)
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

DATAFLOW NAME:	VALID-SECONDARY-I-FRAME-PACKET
ALIASES:	NONE
COMPOSITION:	VALID-SECONDARY-I-FRAME-PACKET = TRANSMITTED-SECONDARY-I-FRAME-PACKET + SECONDARY-I-FRAME-CONTROL-FIELD
NOTES:	EXECUTE SECONDARY I-FRAME PACKET LAYER

DATAFLOW NAME:	WINDOWED-I-FRAME
ALIASES:	NONE
COMPOSITION:	WINDOWED-I-FRAME = LEVEL-3-PACKET + N(R) + N(S) + [POLL-BIT | FINAL-BIT]
NOTES:	WINDOW PRIMARY, SECONDARY INFORMATION BLOCKS LAYER

FILE DEFINITIONS

FILE OR DATABASE NAME:	PENDING-STATE-CHANGE
ALIASES:	NONE
COMPOSITION:	PENDING-STATE-CHANGE = [DISCONNECT-LINK | SETUP-LINK]
ORGANIZATION:	SINGLE VARIABLE
NOTES:	EXECUTE U-FRAME RESPONSE LAYER

FILE OR DATABASE NAME:	PRIMARY-V(R)
ALIASES:	NONE
COMPOSITION:	PRIMARY-V(R) IS THE NEXT PACKET EXPECTED NUMBER IT CAN RANGE FROM 0 TO 7
ORGANIZATION:	SINGLE VARIABLE
NOTES:	EXECUTE SECONDARY I-FRAME PACKET LAYER

FILE OR DATABASE NAME:	PRIMARY-V(S)
ALIASES:	NONE
COMPOSITION:	PRIMARY-V(S) IS AN INTEGER BETWEEN 0 AND 7 THAT INDICATES THE SEQUENCE NUMBER OF THE NEXT I-FRAME TO BE SENT
ORGANIZATION:	SINGLE VARIABLE
NOTES:	WINDOW PRIMARY INFORMATION BLOCKS LAYER

FILE OR DATABASE NAME:	SECONDARY-V(R)
ALIASES:	NONE
COMPOSITION:	SECONDARY-V(R) IS THE NEXT PACKET EXPECTED NUMBER IT CAN RANGE FROM 0 TO 7
ORGANIZATION:	SINGLE VARIABLE
NOTES:	HDLC SECONDARY NODE PROTOCOL LAYER

FILE OR DATABASE NAME:	SECONDARY-V(S)
ALIASES:	NONE
COMPOSITION:	SECONDARY-V(S) IS AN INTEGER BETWEEN 0 AND 7 THAT INDICATES THE SEQUENCE NUMBER OF THE NEXT I-FRAME TO BE SENT
ORGANIZATION:	SINGLE VARIABLE
NOTES:	WINDOW SECONDARY INFORMATION BLOCKS LAYER

PROCESS SPECIFICATIONS

```
PROCESS NAME:          EXTRACT INFORMATION BETWEEN FLAGS
PROCESS NUMBER:        1.1.1
PROCESS DESCRIPTION:
Strip off first SYN (flag character)
While SYN and ABORT not encountered, assemble LEVEL-2-PACKET
If ABORT encountered then discard partially assembled
  LEVEL-2-PACKET as an ABORTED-PACKET
Else output LEVEL-2-PACKET when second SYN is encountered
```

```
PROCESS NAME:          UNSTUFF 0'S
PROCESS NUMBER:        1.1.2
PROCESS DESCRIPTION:
Whenever five consecutive 1's are followed by a 0,
  delete the 0 from the LEVEL-2-PACKET
Output the new packet as an UNSTUFFED-PACKET
```

```
PROCESS NAME:          CHECK LENGTH
PROCESS NUMBER:        1.1.3
PROCESS DESCRIPTION:
If UNSTUFFED-PACKET is less than 32 bits long then it is an
  INVALID-LENGTH-PACKET
Else it is a VALID-LENGTH-PACKET
```

```
PROCESS NAME:          GENERATE CRC REMAINDER
PROCESS NUMBER:        1.1.4
PROCESS DESCRIPTION:
Divide the VALID-LENGTH-PACKET by the CRC polynomial which
 is x**16 + x**12 + x**5 + 1 modulo 2
Multiply the VALID-LENGTH-PACKET by x**16 and then divide it
 by the CRC polynomial
Add the remainders of the above two operations on the
  VALID-LENGTH-PACKET modulo 2 and take the 1's complement
Output the result as the LOCALLY-GENERATED-CRC-POLYNOMIAL
```

```
PROCESS NAME:          CHECK FCS BLOCK
PROCESS NUMBER:        1.1.5
PROCESS DESCRIPTION:
If the FCS-BLOCK and the LOCALLY-GENERATED-CRC-POLYNOMIAL
  are equal then the VALID-LENGTH-PACKET is a VALID-PACKET
Else it is a FCS-ERROR-PACKET
```

```
PROCESS NAME:          DECODE SECONDARY PACKET ADDRESS AND
                        CONTROL FIELDS
PROCESS NUMBER:        1.2
PROCESS DESCRIPTION:
If ADDRESS-FIELD is (10000000) then
  if Bit 1 of CONTROL-FIELD = 0
    then SECONDARY-VALID-PACKET is SECONDARY-TO-PRIMARY-I-FRAME
  else if Bit 2 of CONTROL-FIELD = 0
    then SECONDARY-VALID-PACKET is SECONDARY-TO-PRIMARY-S-
      FRAME-RESPONSE
  else SECONDARY-VALID-PACKET is SECONDARY-TO-PRIMARY-U-FRAME-
      RESPONSE
Else SECONDARY-VALID-PACKET is an INVALID-SECONDARY-PACKET-
  TYPE
```

```
PROCESS NAME:          VALIDATE I-FRAME PACKET
PROCESS NUMBER:        1.3.1
PROCESS DESCRIPTION:
If N(S) of the SECONDARY-I-FRAME-PACKET = PRIMARY-V(R)
  then the SECONDARY-I-FRAME-PACKET is a VALID-SECONDARY-
    I-FRAME-PACKET
Else it is an INVALID-SECONDARY-N(R)-I-FRAME
```

```
PROCESS NAME:          PARSE I-FRAME
PROCESS NUMBER:        1.3.2
PROCESS DESCRIPTION:
Output the first octet of the VALID-SECONDARY-I-FRAME as the
  SECONDARY-I-FRAME-CONTROL-FIELD
Output Bit 5 of the first octet of the VALID-SECONDARY-I-
  FRAME as the I-FRAME-FINAL-BIT
Output all octets following the first octet as the
  TRANSMITTED-SECONDARY-LEVEL-3-PACKET
```

```
PROCESS NAME:          PARSE I-FRAME CONTROL FIELD
PROCESS NUMBER:        1.3.3
PROCESS DESCRIPTION:
Output Bits 2,3,4 as N(S)
Output Bits 6,7,8 as N(R)
```

```
PROCESS NAME:          PARSE S-FRAME RESPONSE CONTROL FIELD
PROCESS NUMBER:        1.4.1
PROCESS DESCRIPTION:
RESPONSE-SUPERVISORY-FUNCTION-BITS = Bits 3,4,6,7,8 of
  S-FRAME-RESPONSE-CONTROL-FIELD
S-FRAME-FINAL-BIT = Bit 5 of S-FRAME-CONTROL-FIELD


PROCESS NAME:          DECODE RESPONSE SUPERVISORY BITS
PROCESS NUMBER:        1.4.2
PROCESS DESCRIPTION:
Case Bits 3,4 of

  00:  RR-RESPONSE
  10:  RNR-RESPONSE
  01:  REJ-RESPONSE
  11:  INVALID-S-FRAME-RESPONSE


PROCESS NAME:          SEND READY STATUS AND EXTRACT N(R)
PROCESS NUMBER:        1.4.3
PROCESS DESCRIPTION:
Output READY-INDICATION
Set READY-N(R) = Bits 6,7,8 of RR-RESPONSE


PROCESS NAME:          SEND BUSY STATUS AND EXTRACT N(R)
PROCESS NUMBER:        1.4.4
PROCESS DESCRIPTION:
Output BUSY-INDICATION
Set BUSY-N(R) = Bits 6,7,8 of RNR-RESPONSE


PROCESS NAME:          SEND REJECT CONDITION AND EXTRACT N(R)
PROCESS NUMBER:        1.4.5
PROCESS DESCRIPTION:
Output REJECTION-EXCEPTION-CONDITION
Set REJECT-N(R) = Bits 6,7,8 of REJ-RESPONSE


PROCESS NAME:          PARSE U-FRAME RESPONSE CONTROL FIELD
PROCESS NUMBER:        1.5.1
PROCESS DESCRIPTION:
RESPONSE-MODIFIER-BITS = Bits 3,4,6,7,8 of the U-FRAME-
                                   RESPONSE
U-FRAME-FINAL-BIT = Bit 5 of the U-FRAME-RESPONSE
```

```
PROCESS NAME:          DECODE RESPONSE MODIFIER BITS
PROCESS NUMBER:        1.5.2
PROCESS DESCRIPTION:
Case Bits 3,4,6,7,8 of
  (11000):  Output DM-RESPONSE
  (10001):  Output CMDR(FRMR)-RESPONSE
  (00110):  Output UA-RESPONSE
Otherwise:  Output INVALID-U-FRAME-RESPONSE
```

```
PROCESS NAME:          SET UP LINK
PROCESS NUMBER:        1.5.3
PROCESS DESCRIPTION:
If DM-RESPONSE or TRANSMITTED-SECONDARY-CMDR(FRMR)-RESPONSE
  is received then output an SABM-COMMAND and set the
  PENDING-STATE-CHANGE to LINK-SETUP state
```

```
PROCESS NAME:          EXECUTE PENDING STATE CHANGE
PROCESS NUMBER:        1.5.4
PROCESS DESCRIPTION:
If UA-RESPONSE is received then set LINK-STATUS to PENDING-
  STATE-CHANGE
```

```
PROCESS NAME:          RESET V(S)
PROCESS NUMBER:        1.6.1
PROCESS DESCRIPTION:
When a REJECTION-EXCEPTION-CONDITION is received
  Update PRIMARY-V(S) to equal the REJECT-N(R)
  Output a REJ-READY-INDICATION
```

```
PROCESS NAME:          UPDATE LOWER WINDOW EDGE
PROCESS NUMBER:        1.6.2
PROCESS DESCRIPTION:
Set UPDATED-LOWER-WINDOW-EDGE = N(R)
```

```
PROCESS NAME:          POLL SECONDARY FOR READY STATE
PROCESS NUMBER:        1.6.3
PROCESS DESCRIPTION:
If BUSY-INDICATION is received then after POLL-DELAY output
  an RR-COMMAND
```

```
PROCESS NAME:          PLACE I-FRAME IN X.25 LINK QUEUE
PROCESS NUMBER:        1.6.4
PROCESS DESCRIPTION:
While LINK-STATUS is in READY state
  and PRIMARY-V(S) < UPDATED-LOWER-WINDOW-EDGE - 1
  and no REJ-READY-INDICATION is present
        Send PRIMARY-LEVEL-3-PACKET as WINDOWED-I-FRAME
          with N(R) = PRIMARY-V(R)
               N(S) = PRIMARY-V(S)
        Increment PRIMARY-V(S)
If REJ-READY-INDICATION then
        Retransmit PRIMARY-LEVEL-3-PACKETs starting with the
           WINDOWED-I-FRAME with N(S) = PRIMARY-V(S)
        Increment PRIMARY-V(S)
```

```
PROCESS NAME:          SELECT PACKET FOR X.25 LINK
PROCESS NUMBER:        1.6.5
PROCESS DESCRIPTION:
Select packets from the transmit queue according to the
  following priority
     Priority 1:   SABM-COMMAND or RESET-COMMAND-PACKET
     Priority 2:   RR-COMMAND
     Priority 3:   WINDOWED-I-FRAME
If the queue is empty for MAX-TIME-BETWEEN-UPDATES then send
   an RR-RESPONSE with N(R) = PRIMARY-N(R) to update
   SECONDARY NODE LOWER-WINDOW-EDGE
Generate FCS-BLOCK
   Divide the VALID-LENGTH-PACKET by the CRC polynomial which
    is x**16 + x**12 + x**5 + 1 modulo 2
   Multiply the VALID-LENGTH-PACKET by x**16 and then divide
    it by the CRC polynomial
   Add the remainders of the above two operations on the
     VALID-LENGTH-PACKET modulo 2 and take the 1's complement
   The result is the FCS-BLOCK
Append the FCS-BLOCK to the packet to be transmitted and
  output the combination as a PRIMARY-LEVEL-2-PACKET
```

```
PROCESS NAME:          RECOVER FROM PROCEDURE ERROR
PROCESS NUMBER:        1.7
PROCESS DESCRIPTION:
If an INVALID-LENGTH-SECONDARY-PACKET or an INVALID-
 SECONDARY-PACKET-TYPE or an INVALID-SECONDARY-N(R)-I-FRAME
 then reset the link using an SABM-COMMAND
```

PROCESS NAME: SEND PACKET
PROCESS NUMBER: 1.8
PROCESS DESCRIPTION:
Send continuous SYNs while there are no LEVEL-2-PACKETs to transmit
When a LEVEL-2-PACKET is available, add a SYN to the front of the packet and another SYN to the rear of the packet and insert it into the outgoing bit stream

PROCESS NAME: TRANSMIT BIT STREAMS
PROCESS NUMBER: 2
PROCESS DESCRIPTION:
Use the Physical Level Protocol to transmit the PRIMARY-OUTGOING-BIT-STREAM to the SECONDARY NODE and the SECONDARY-OUTGOING-BIT-STREAM to the PRIMARY NODE

PROCESS NAME: EXTRACT VALID PACKET
PROCESS NUMBER: 3.1
PROCESS DESCRIPTION: SEE PROCESSES 1.1.1 THROUGH 1.1.5

PROCESS NAME: DECODE PRIMARY PACKET ADDRESS AND CONTROL FIELDS
PROCESS NUMBER: 3.2
PROCESS DESCRIPTION:

```
If ADDRESS-FIELD is (10000000) then
  if Bit 1 of CONTROL-FIELD = 0
    then VALID-PRIMARY-PACKET is an PRIMARY-TO-SECONDARY-I-
       FRAME-PACKET
  else if Bit 2 of CONTROL-FIELD = 0 then VALID-PRIMARY-
       PACKET is an S-FRAME-COMMAND
  else VALID-PRIMARY-PACKET is a U-FRAME-COMMAND
Else VALID-PRIMARY-PACKET is an INVALID-PRIMARY-PACKET-TYPE
```

PROCESS NAME: VALIDATE I-FRAME PACKET
PROCESS NUMBER: 3.3.1
PROCESS DESCRIPTION:

```
If N(S) of the PRIMARY-I-FRAME-PACKET = SECONDARY-V(R)
  then the PRIMARY-I-FRAME-PACKET is a VALID-PRIMARY-
    I-FRAME-PACKET
Else it is an OUT-OF-SEQUENCE-PRIMARY-I-FRAME
```

PROCESS NAME: PARSE I-FRAME
PROCESS NUMBER: 3.3.2
PROCESS DESCRIPTION:
Output the first octet of the VALID-PRIMARY-I-FRAME as the PRIMARY-I-FRAME-CONTROL-FIELD
Output Bit 5 of the first octet of the VALID-PRIMARY-I-FRAME as the I-FRAME-POLL-BIT
Output all octets following the first octet as the TRANSMITTED-PRIMARY-NODE-LEVEL-3-PACKET

PROCESS NAME: PARSE I-FRAME CONTROL FIELD
PROCESS NUMBER: 3.3.3
PROCESS DESCRIPTION:
Output Bits 2,3,4 as N(S)
Output Bits 6,7,8 as N(R)

PROCESS NAME: PARSE S-FRAME COMMAND
PROCESS NUMBER: 3.4.1
PROCESS DESCRIPTION:
S-FRAME-CONTROL-FIELD = First octet of S-FRAME-COMMAND
S-FRAME-POLL-BIT = Bit 5 of S-FRAME-CONTROL-FIELD

PROCESS NAME: PARSE S-FRAME CONTROL FIELD
PROCESS NUMBER: 3.4.2
PROCESS DESCRIPTION:
Case Bits 3,4 of

00:	RR-COMMAND
10:	RNR-COMMAND
01:	REJ-COMMAND
11:	INVALID-S-FRAME-COMMAND

PROCESS NAME: SEND READY STATUS AND EXTRACT N(R)
PROCESS NUMBER: 3.4.3
PROCESS DESCRIPTION:
Output READY-INDICATION
Set PRIMARY-READY-N(R) = Bits 6,7,8 of RR-COMMAND

PROCESS NAME: SEND BUSY STATUS AND EXTRACT N(R)
PROCESS NUMBER: 3.4.4
PROCESS DESCRIPTION:
Output PRIMARY-BUSY-INDICATION
Set BUSY-PRIMARY-N(R) = Bits 6,7,8 of RNR-COMMAND

```
PROCESS NAME:          SEND REJECT CONDITION AND EXTRACT N(R)
PROCESS NUMBER:        3.4.5
PROCESS DESCRIPTION:
Output PRIMARY-REJECTION-CONDITION
Set REJECT-PRIMARY-N(R) = Bits 6,7,8 of REJ-COMMAND


PROCESS NAME:          PARSE U-FRAME COMMAND CONTROL FIELD
PROCESS NUMBER:        3.5.1
PROCESS DESCRIPTION:
COMMAND-MODIFIER-BITS = Bits 3,4,6,7,8 of the U-FRAME-
                            COMMAND
U-FRAME-POLL-BIT = Bit 5 of the U-FRAME-COMMAND


PROCESS NAME:          DECODE COMMAND MODIFIER BITS
PROCESS NUMBER:        3.5.2
PROCESS DESCRIPTION:
Case Bits 3,4,6,7,8 of
  (11100):  Output TRANSMITTED-SABM-COMMAND
  (00010):  Output DISC-COMMAND
Otherwise:  Output INVALID-U-FRAME-COMMAND


PROCESS NAME:          DISCONNECT LINK
PROCESS NUMBER:        3.5.3
PROCESS DESCRIPTION:
Upon receipt of a DISC-COMMAND
  If not in Disconnected state then
   Set SECONDARY NODE state to Disconnected state
   Output DISC-UA-RESPONSE
  Else output DM-RESPONSE


PROCESS NAME:          SET UP LINK
PROCESS NUMBER:        3.5.4
PROCESS DESCRIPTION:
Upon receipt of a TRANSMITTED-SABM-COMMAND
   Reset SECONDARY-V(R) and SECONDARY-V(S) to 0
   Set SECONDARY NODE state to Ready
   Output SABM-UA-RESPONSE
```

```
PROCESS NAME:          RESET V(S)
PROCESS NUMBER:        3.6.1
PROCESS DESCRIPTION:
When a PRIMARY-REJECTION-CONDITION is received
  Update SECONDARY-V(S) to equal the PRIMARY-REJECT-N(R)
  Output a REJ-READY-INDICATION
When a SABM-UA-RESPONSE is received reset SECONDARY-V(S) to 0
```

```
PROCESS NAME:          UPDATE LOWER WINDOW EDGE
PROCESS NUMBER:        3.6.2
PROCESS DESCRIPTION:
Set UPDATED-LOWER-WINDOW-EDGE = N(R)
```

```
PROCESS NAME:          SET RESPONSE FINAL BIT
PROCESS NUMBER:        3.6.3
PROCESS DESCRIPTION:
If POLL-BIT = 1 then set SECONDARY-FINAL-BIT = 1
```

```
PROCESS NAME:          PLACE I-FRAME IN X.25 LINK
PROCESS NUMBER:        3.6.4
PROCESS DESCRIPTION:
While SECONDARY NODE is in Ready state
  and SECONDARY-V(S) < UPDATED-LOWER-WINDOW-EDGE - 1
  and no REJ-READY-INDICATION is present
        Send SECONDARY-LEVEL-3-PACKET as WINDOWED-I-FRAME
          with N(R) = SECONDARY-V(R)
               N(S) = SECONDARY-V(S)
        Increment SECONDARY-V(S)
If REJ-READY-INDICATION then
        Retransmit SECONDARY-LEVEL-3-PACKETs starting with the
           WINDOWED-I-FRAME with N(S) = SECONDARY-V(S)
        Increment SECONDARY-V(S)
```

```
PROCESS NAME:          SELECT PACKET FOR X.25 LINK
PROCESS NUMBER:        3.6.5
PROCESS DESCRIPTION:
Select packets from the transmit queue according to the
  following priority
     Priority 1:   CMDR(FRMR)-RESPONSE
     Priority 2:   DISC-UA-REPONSE or DM-RESPONSE
     Priority 3:   RR-RESPONSE or RNR-RESPONSE
     Priority 4:   WINDOWED-I-FRAME
If the queue is empty for MAX-TIME-BETWEEN-UPDATES then send
   an RR-RESPONSE with N(R) = SECONDARY-N(R) to update
   PRIMARY NODE LOWER-WINDOW-EDGE
```

```
Generate FCS-BLOCK
   Divide the VALID-LENGTH-PACKET by the CRC polynomial which
    is x**16 + x**12 + x**5 + 1 modulo 2
   Multiply the VALID-LENGTH-PACKET by x**16 and then divide
    it by the CRC polynomial
   Add the remainders of the above two operations on the
     VALID-LENGTH-PACKET modulo 2 and take the 1's complement
   The result is the FCS-BLOCK
Append the FCS-BLOCK to the packet to be transmitted
Set FINAL-BIT if SECONDARY-FINAL-BIT is set
Output a SECONDARY-LEVEL-2-PACKET
```

```
PROCESS NAME:          RESPOND TO STATUS REQUEST
PROCESS NUMBER:        3.6.6
PROCESS DESCRIPTION:
If POLL-BIT is set then
  if SECONDARY NODE is in Ready state then output RR-RESPONSE
  else output RNR-RESPONSE
```

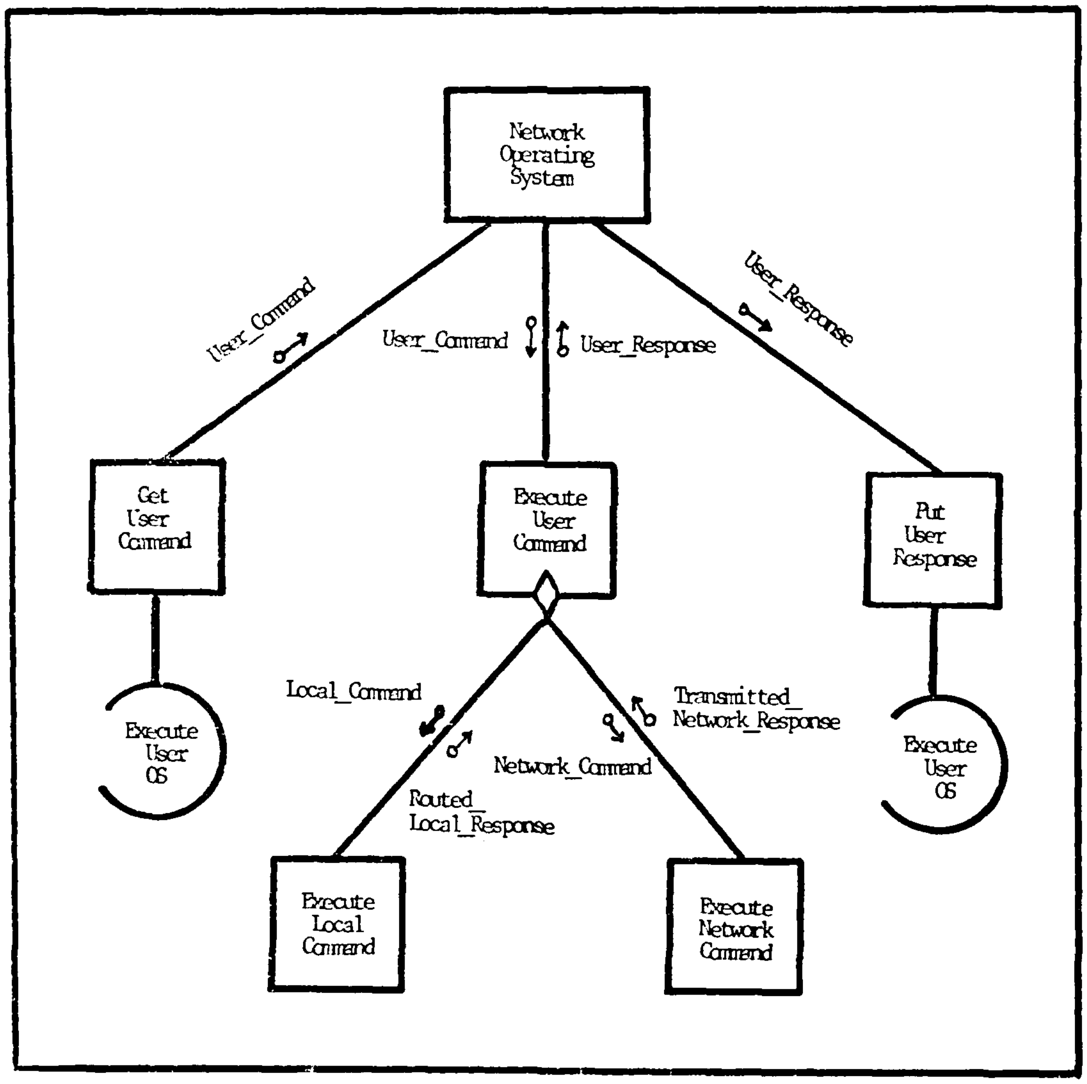
```
PROCESS NAME:          REQUEST RECOVERY FROM PROCEDURE ERROR
PROCESS NUMBER:        3.7
PROCESS DESCRIPTION:
If an INVALID-LENGTH-PRIMARY-PACKET or an INVALID-PRIMARY-
  PACKET-TYPE or an OUT-OF-SEQUENCE-PRIMARY-I-FRAME-PACKET
  is received then output a CMDR(FRMR)-RESPONSE
```

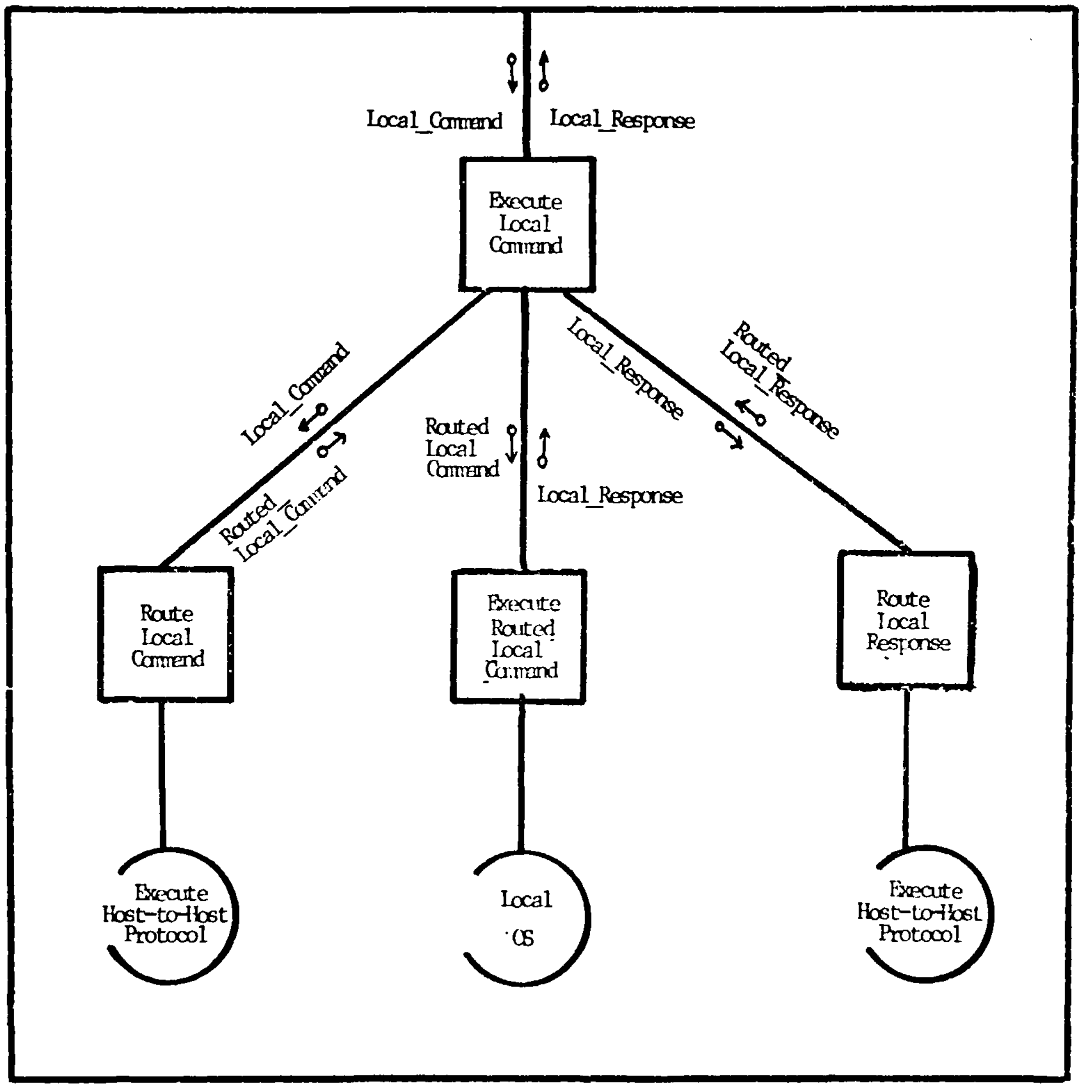
```
PROCESS NAME:          SEND PACKET
PROCESS NUMBER:        3.8
PROCESS DESCRIPTION:   SEE PROCESS 1.8
```

Appendix D

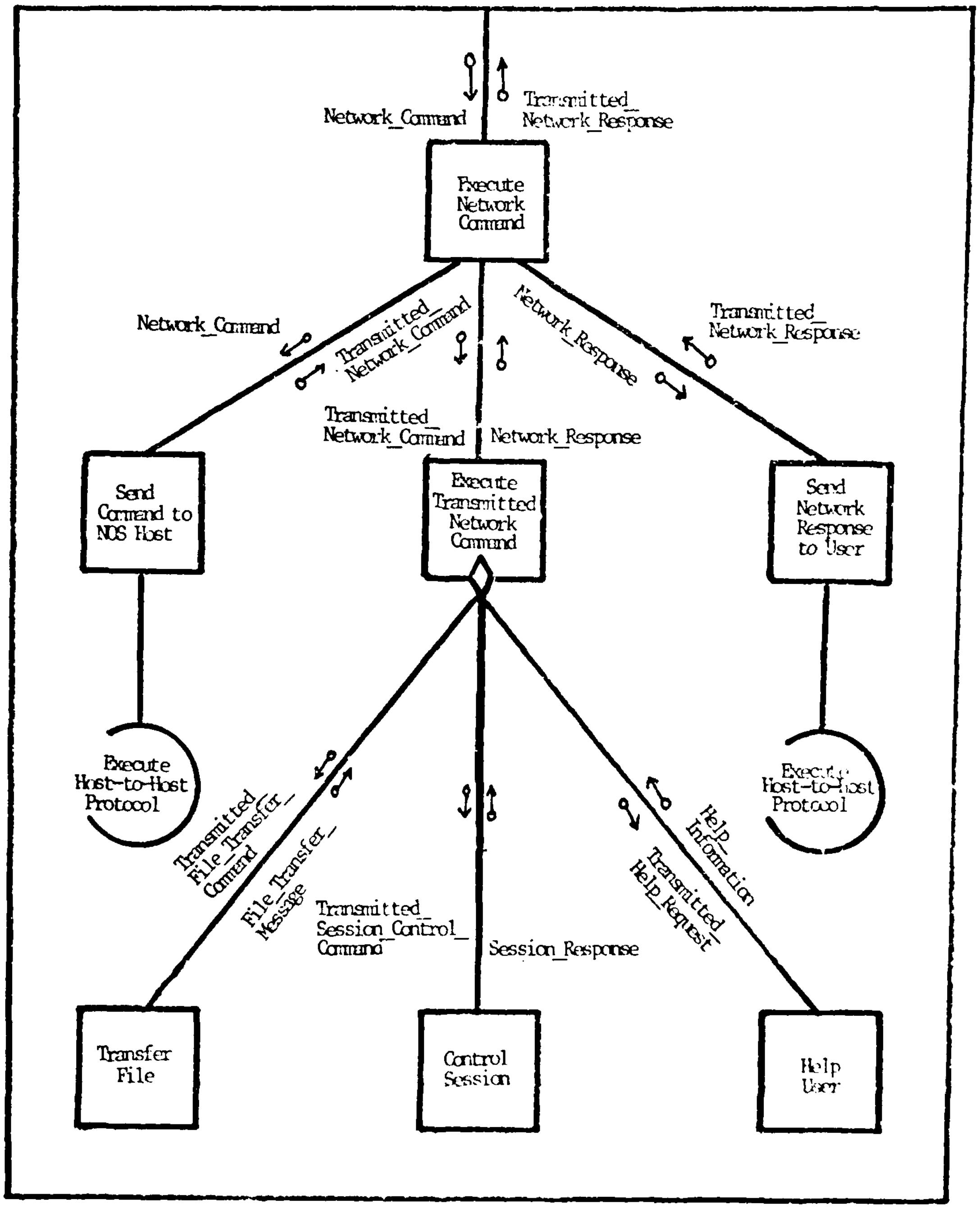
Module Structure Charts

This appendix contains the complete set of module structure charts needed to specify the design of the software for DELNET. These charts were drawn using Yourdon and Constantine's structured design techniques (Ref. 22).


Network Operating System
User_Command
User_Command
User_Response
User_Response
Get User Command
Execute User Command
Put User Response
Execute User OS
Local_Command
Transmitted_ Network_Response
Network_Command
Routed_ Local_Response
Execute User OS
Execute Local Command
Execute Network Command



Local_Command
Local_Response
Execute Local Command
Local_Command
Routed Local_Command
Routed Local Command
Local_Response
Local_Response
Routed Local_Response
Route Local Command
Execute Routed Local Command
Route Local Response
Execute Host-to-Host Protocol
Local OS
Execute Host-to-Host Protocol


Network_Command
Transmitted_Network_Response
Execute Network Command
Network_Command
Transmitted_Network_Command
Network_Response
Transmitted_Network_Response
Transmitted_Network_Command
Network_Response
Send Command to NOS Host
Execute Transmitted Network Command
Send Network Response to User
Execute Host-to-Host Protocol
Execute Host-to-Host Protocol
Transmitted_File_Transfer_Command
File_Transfer_Message
Transmitted_Session_Control_Command
Session_Response
Help_Information
Transmitted_Help_Request
Transfer File
Control Session
Help User

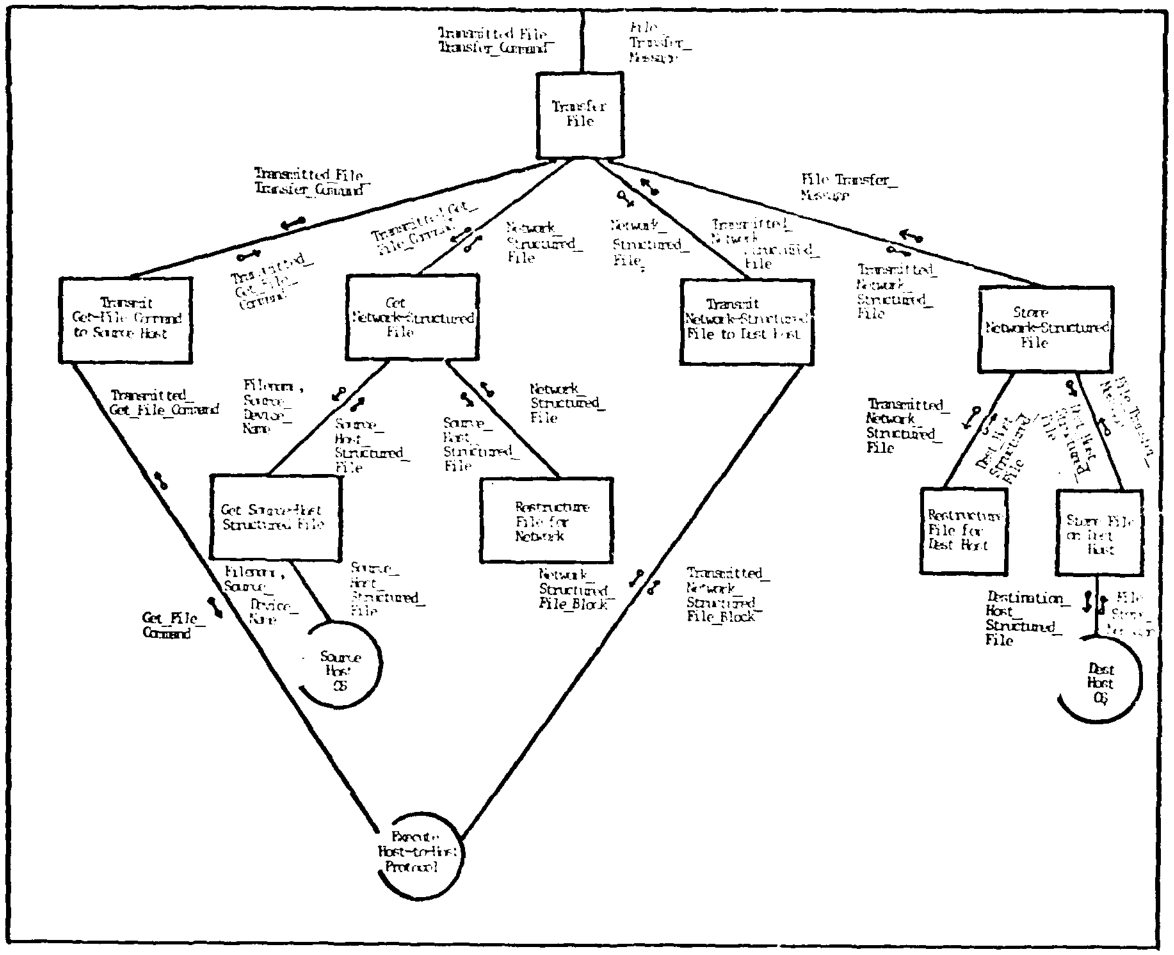
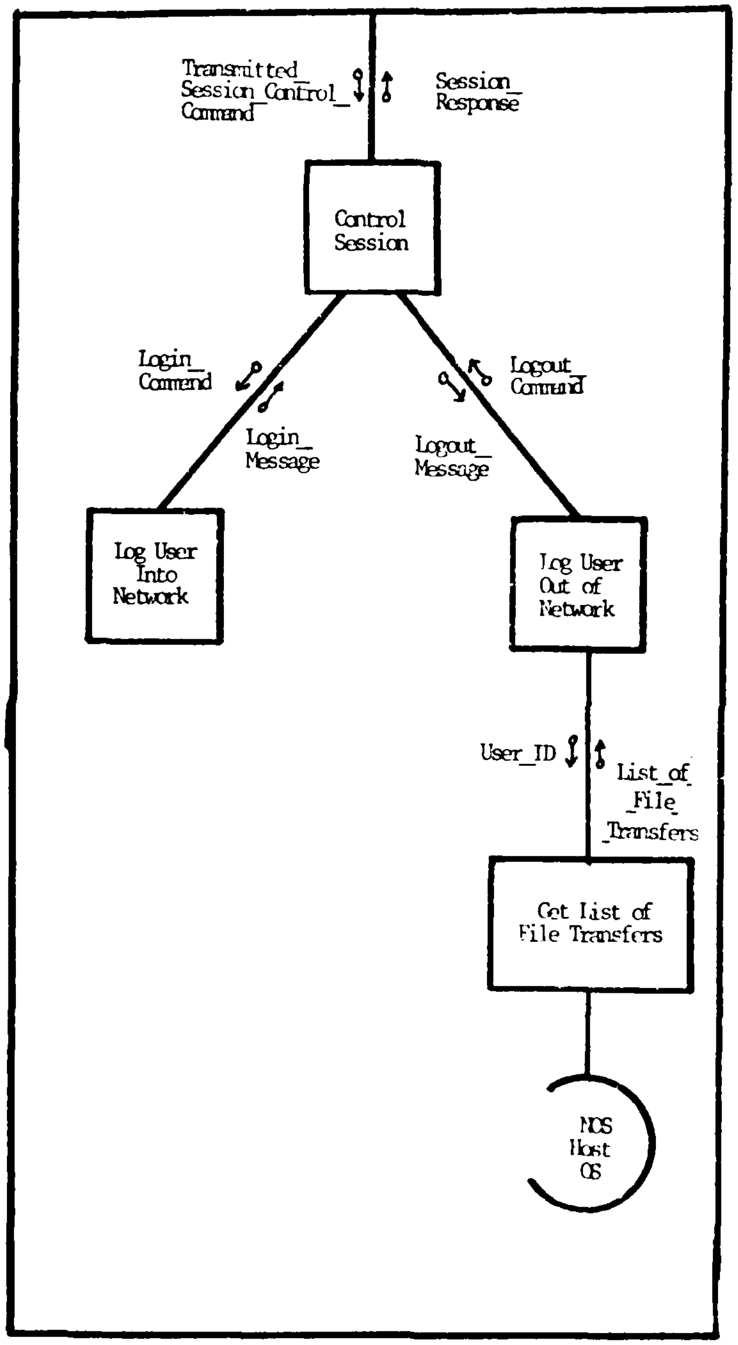

Transmitted_File_Transfer_Command
File_Transfer_Message
Transfer File
Transmitted_File_Transfer_Command
Transmitted_Get_File_Command
Network_Structured_File
Network_Structured_File
Transmitted_Network_Structured_File
File_Transfer_Message
Transmitted_Get_File_Command
Transmitted_Network_Structured_File
Transmit Get-File Command to Source Host
Get Network-Structured File
Transmit Network-Structured File to Dest Host
Store Network-Structured File
Transmitted_Get_File_Command
Filename, Source_Device_Name
Source_Host_Structured_File
Source_Host_Structured_File
Network_Structured_File
Transmitted_Network_Structured_File
Dest_Host_Structured_File
Dest_Host_Structured_File
File_Transfer_Message
Get Source-Host Structured File
Restructure File for Network
Restructure File for Dest Host
Store File on Dest Host
Filename, Source_Device_Name
Source_Host_Structured_File
Network_Structured_File_Block
Transmitted_Network_Structured_File_Block
Destination_Host_Structured_File
File_Store_Message
Get_File_Command
Source Host OS
Dest Host OS
Execute Host-to-Host Protocol

Figure 39. Transfer File Module Structure Chart


Transmitted_
Session_Control_
Command
Session_
Response
Control
Session
Login_
Command
Login_
Message
Logout_
Command
Logout_
Message
Log User
Into
Network
Log User
Out of
Network
User_ID
List_of
_File
_Transfers
Get List of
File Transfers
NOS
Host
OS

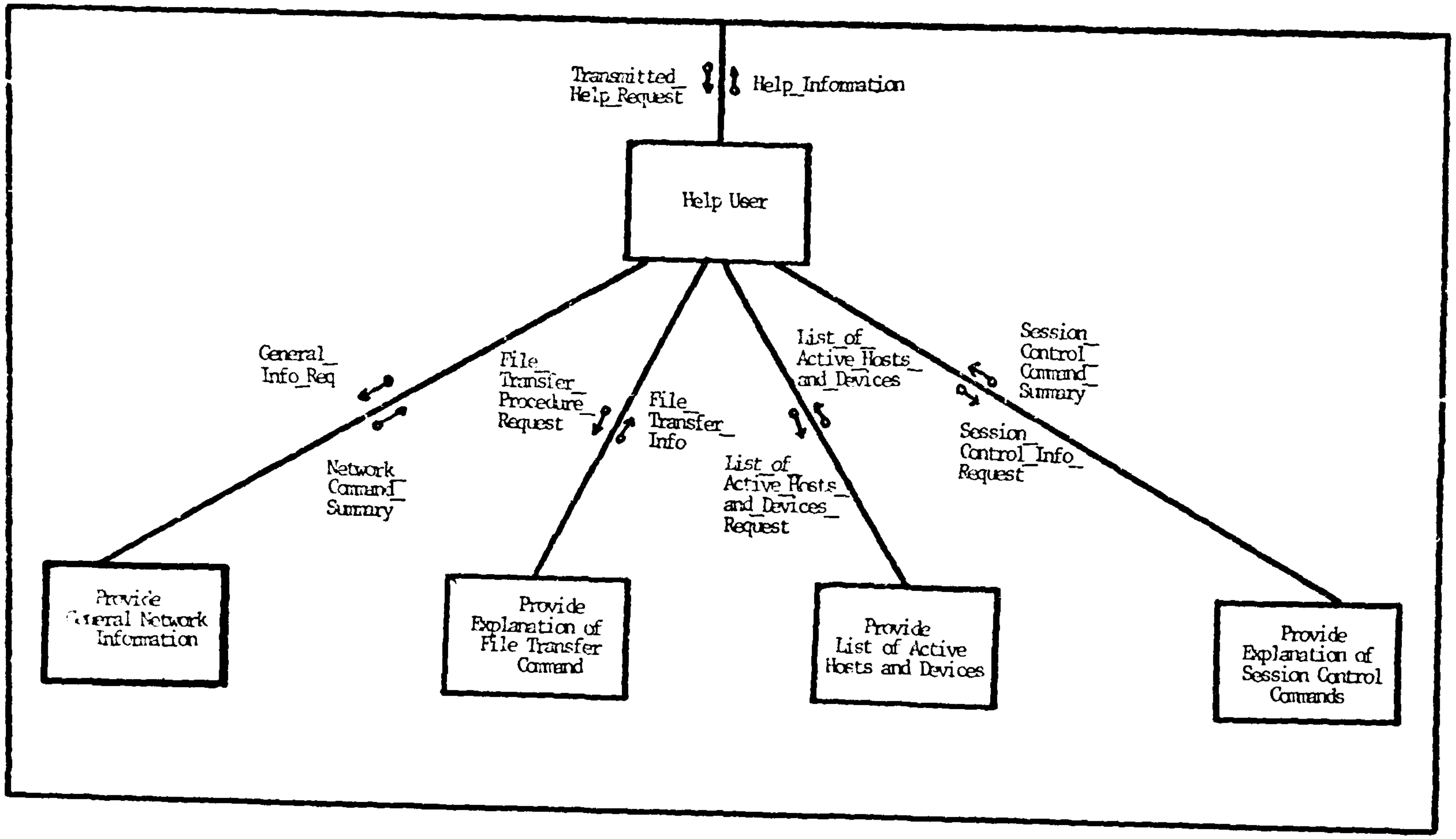
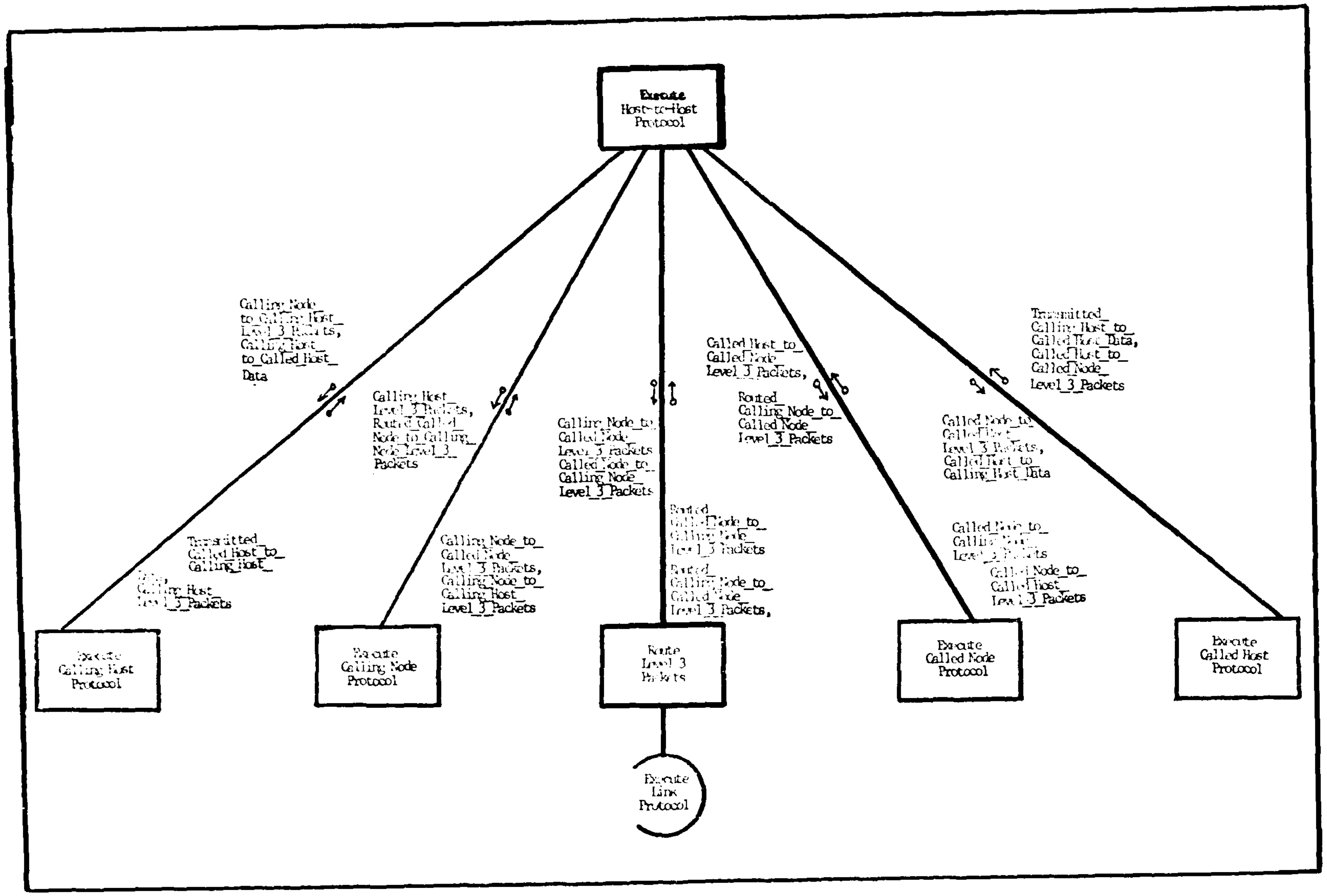
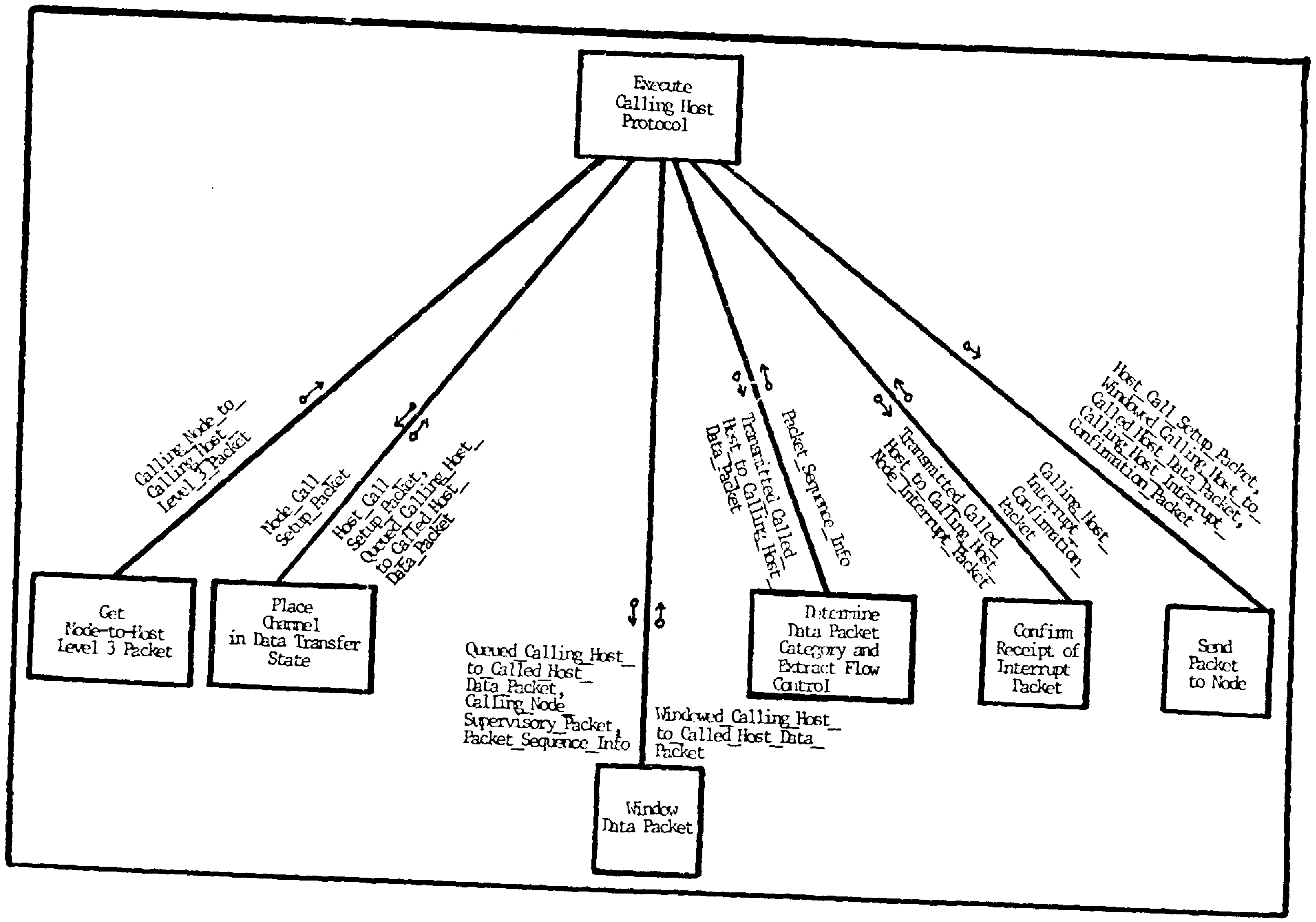
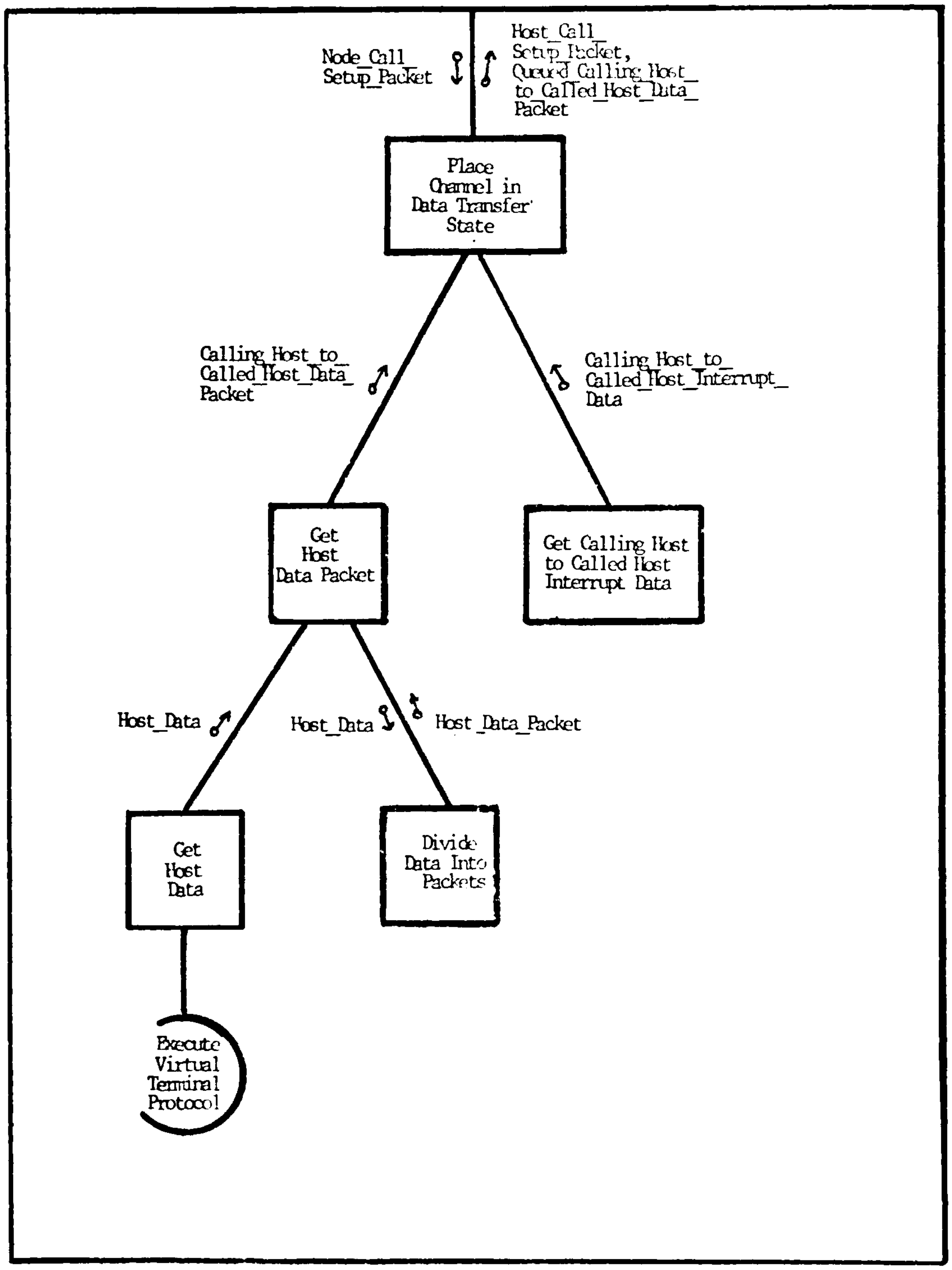

Transmitted_ Help_Request
Help_Information
Help User
General_ Info_Req
Network_ Command_ Summary
File_ Transfer_ Procedure_ Request
File_ Transfer_ Info
List_of_ Active_Hosts_ and_Devices
List_of_ Active_Hosts_ and_Devices_ Request
Session_ Control_ Command_ Summary
Session_ Control_Info_ Request
Provide General Network Information
Provide Explanation of File Transfer Command
Provide List of Active Hosts and Devices
Provide Explanation of Session Control Commands

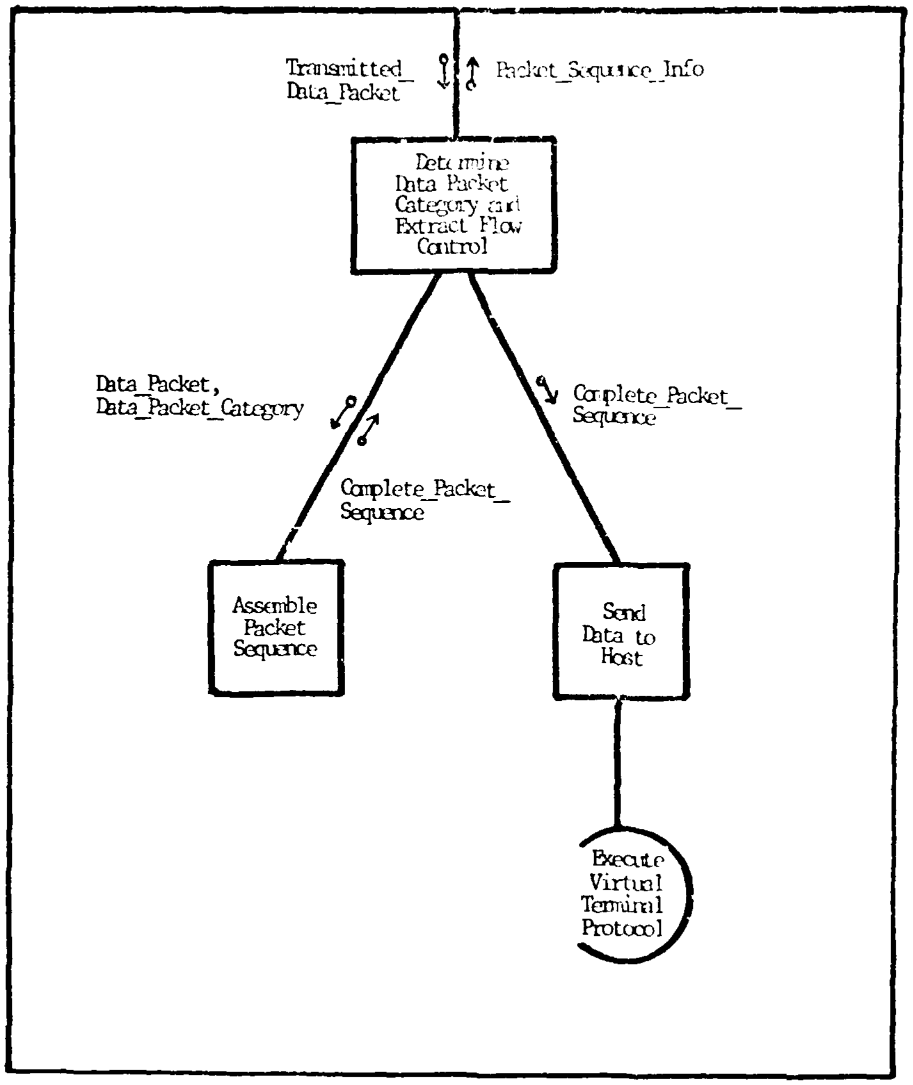
Figure 41. Help User Module Structure Chart


Execute Host-to-Host Protocol
Calling_Node_to_Calling_Host_Level_3_Packets, Calling_Host_to_Called_Host_Data
Transmitted_Called_Host_to_Calling_Host_Data, Calling_Host_Level_3_Packets
Execute Calling Host Protocol
Calling_Host_Level_3_Packets, Routed_Called_Node_to_Calling_Node_Level_3_Packets
Calling_Node_to_Called_Node_Level_3_Packets, Calling_Node_to_Calling_Host_Level_3_Packets
Execute Calling Node Protocol
Calling_Node_to_Called_Node_Level_3_Packets, Called_Node_to_Calling_Node_Level_3_Packets
Routed_Called_Node_to_Calling_Node_Level_3_Packets
Routed_Calling_Node_to_Called_Node_Level_3_Packets,
Route Level 3 Packets
Execute Link Protocol
Called_Host_to_Called_Node_Level_3_Packets,
Routed_Calling_Node_to_Called_Node_Level_3_Packets
Called_Node_to_Calling_Node_Level_3_Packets
Called_Node_to_Called_Host_Level_3_Packets
Execute Called Node Protocol
Transmitted_Calling_Host_to_Called_Host_Data, Called_Host_to_Called_Node_Level_3_Packets
Called_Node_to_Called_Host_Level_3_Packets, Called_Host_to_Calling_Host_Data
Execute Called Host Protocol

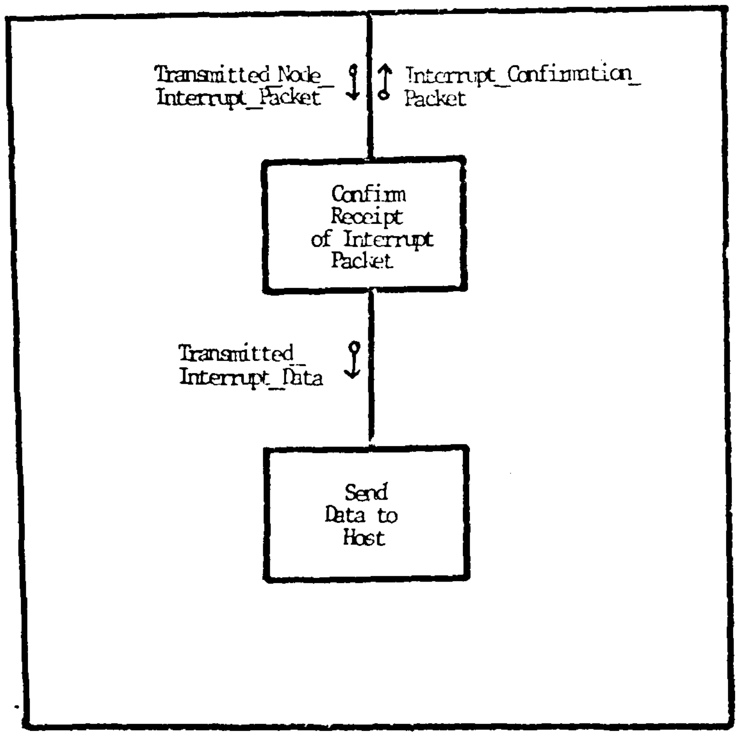

Execute Calling Host Protocol
Get Node-to-Host Level 3 Packet
Calling_Node_to_ Calling_Host_ Level_3_Packet
Place Channel in Data Transfer State
Node_Call_ Setup_Packet
Host_Call_ Setup_Packet, Queued_Calling_Host_ to_Called_Host_ Data_Packet
Window Data Packet
Queued_Calling_Host_ to_Called_Host_ Data_Packet, Calling_Node_ Supervisory_Packet, Packet_Sequence_Info
Windowed_Calling_Host_ to_Called_Host_Data_ Packet
Determine Data Packet Category and Extract Flow Control
Transmitted_Called_ Host_to_Calling_Host_ Data_Packet
Packet_Sequence_Info
Confirm Receipt of Interrupt Packet
Transmitted_Called_ Host_to_Calling_Host_ Node_Interrupt_Packet
Calling_Host_ Interrupt_ Confirmation_ Packet
Send Packet to Node
Host_Call_Setup_Packet, Windowed_Calling_Host_to_ Called_Host_Data_Packet, Calling_Host_Interrupt_ Confirmation_Packet

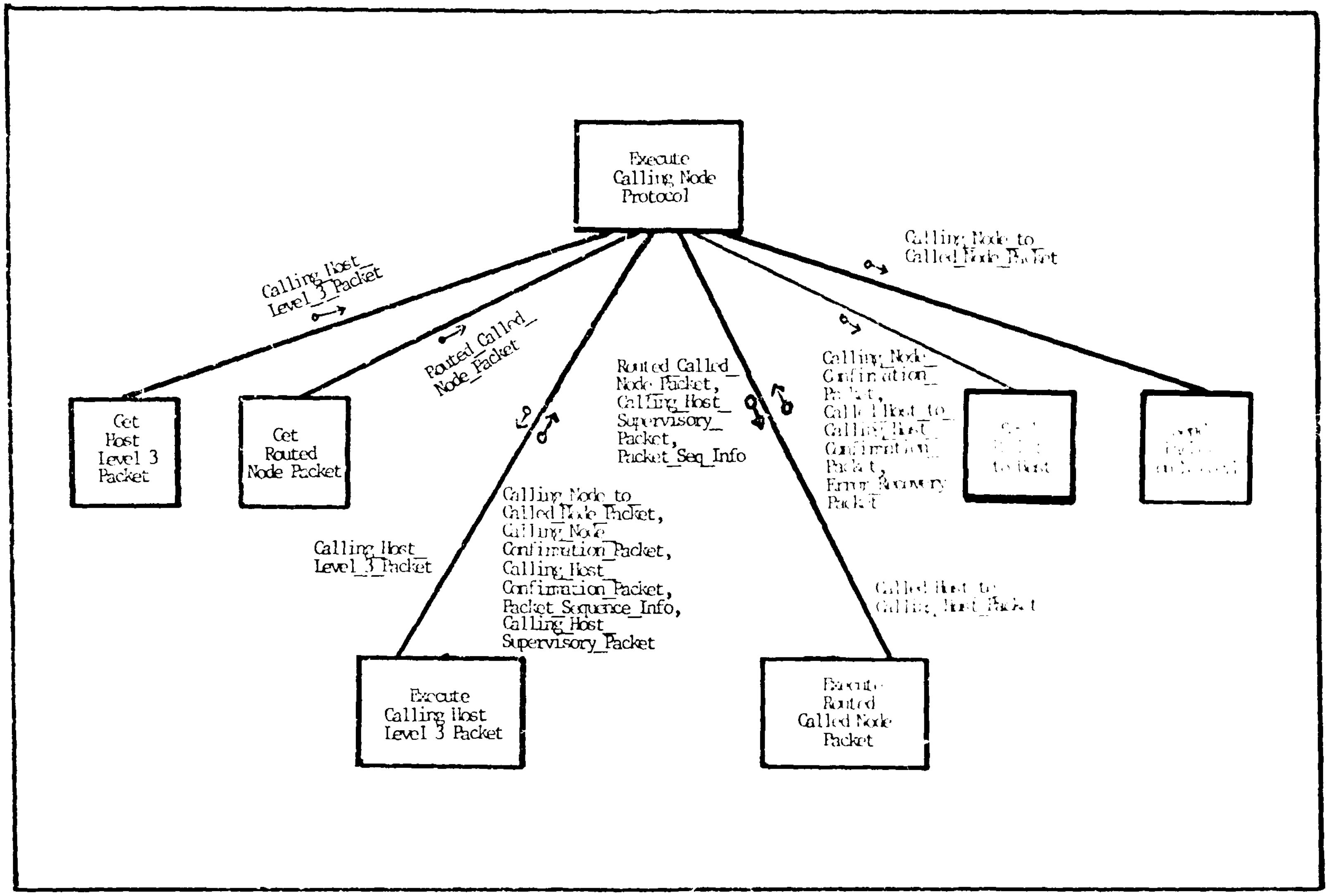


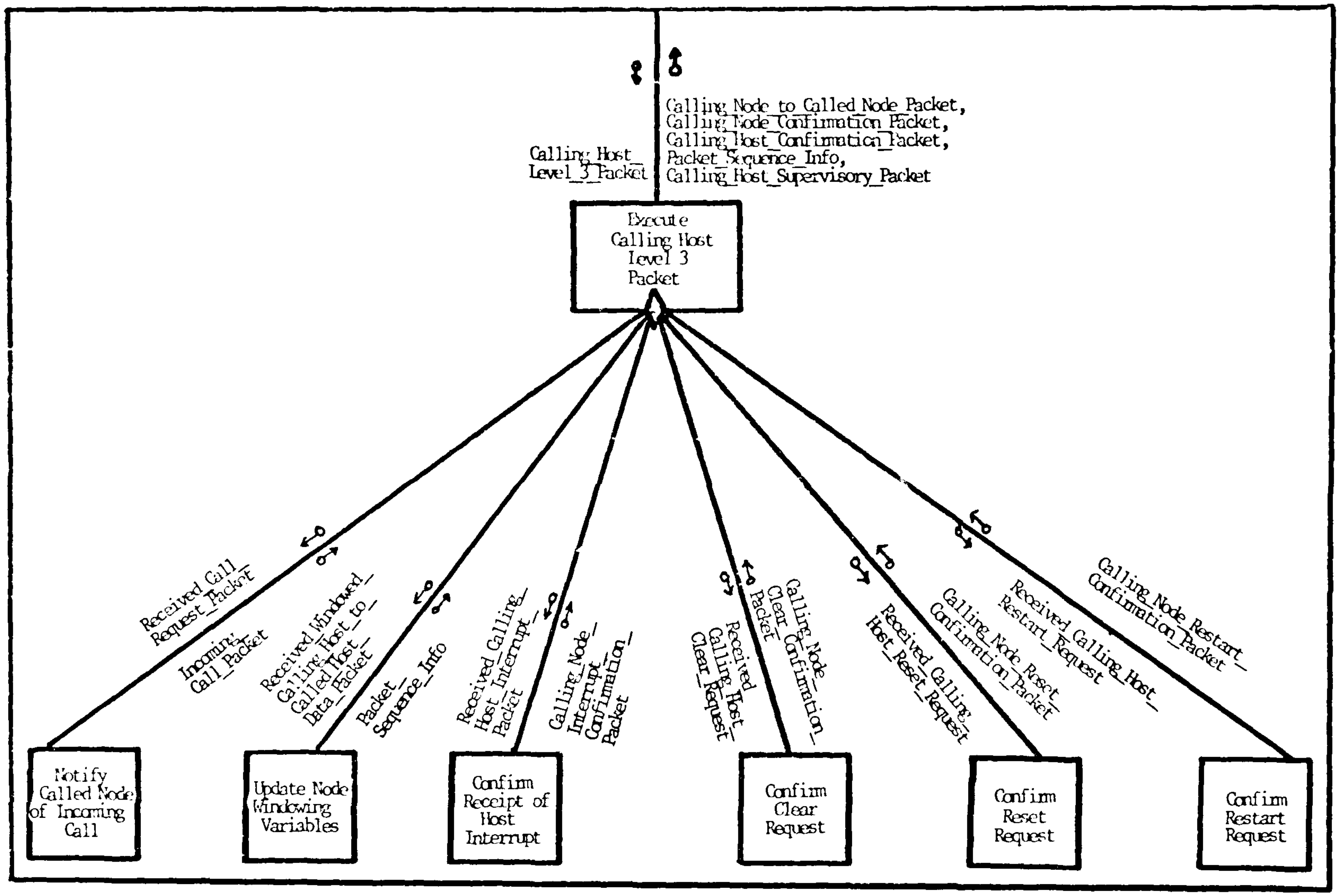
Node_Call_Setup_Packet
Host_Call_Setup_Packet, Queued Calling_Host_to_Called_Host_Data_Packet
Place Channel in Data Transfer State
Calling_Host_to_Called_Host_Data_Packet
Calling_Host_to_Called_Host_Interrupt_Data
Get Host Data Packet
Get Calling Host to Called Host Interrupt Data
Host_Data
Host_Data
Host_Data_Packet
Get Host Data
Divide Data Into Packets
Execute Virtual Terminal Protocol



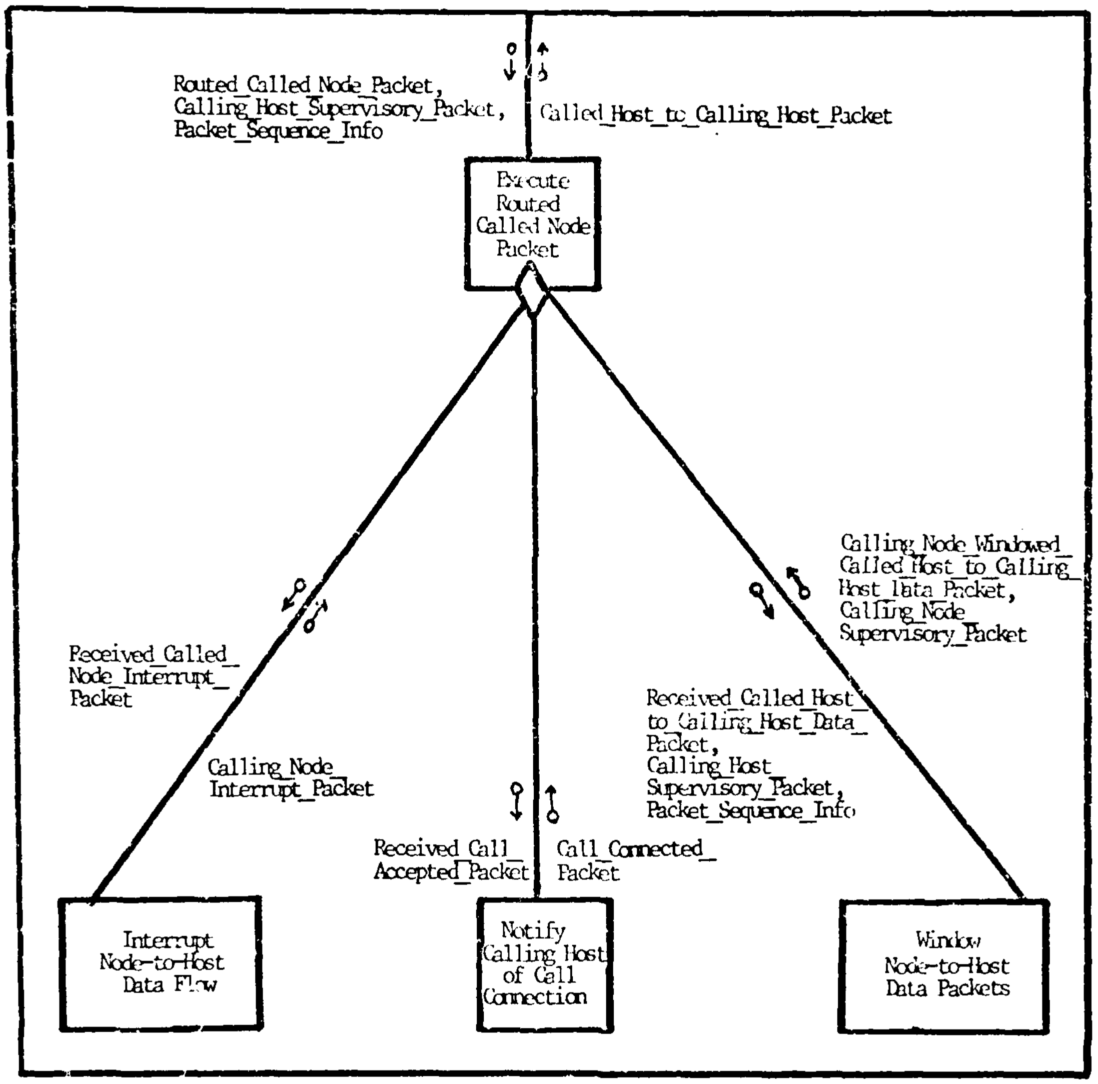
Transmitted_Data_Packet
Packet_Sequence_Info
Determine Data Packet Category and Extract Flow Control
Data_Packet, Data_Packet_Category
Complete_Packet_Sequence
Complete_Packet_Sequence
Assemble Packet Sequence
Send Data to Host
Execute Virtual Terminal Protocol

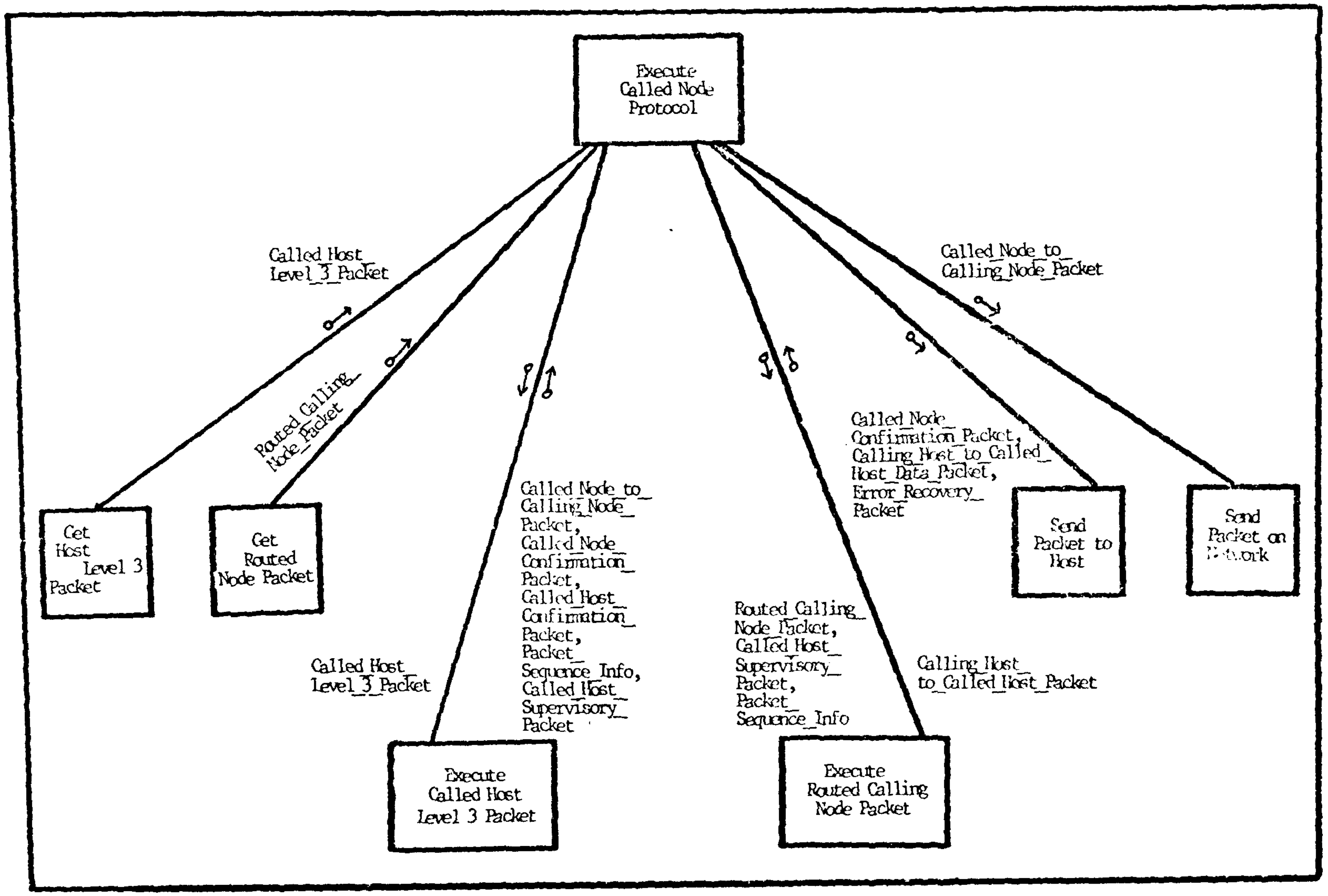

Transmitted_Node_Interrupt_Packet
Interrupt_Confirmation_Packet
Confirm Receipt of Interrupt Packet
Transmitted_Interrupt_Data
Send Data to Host

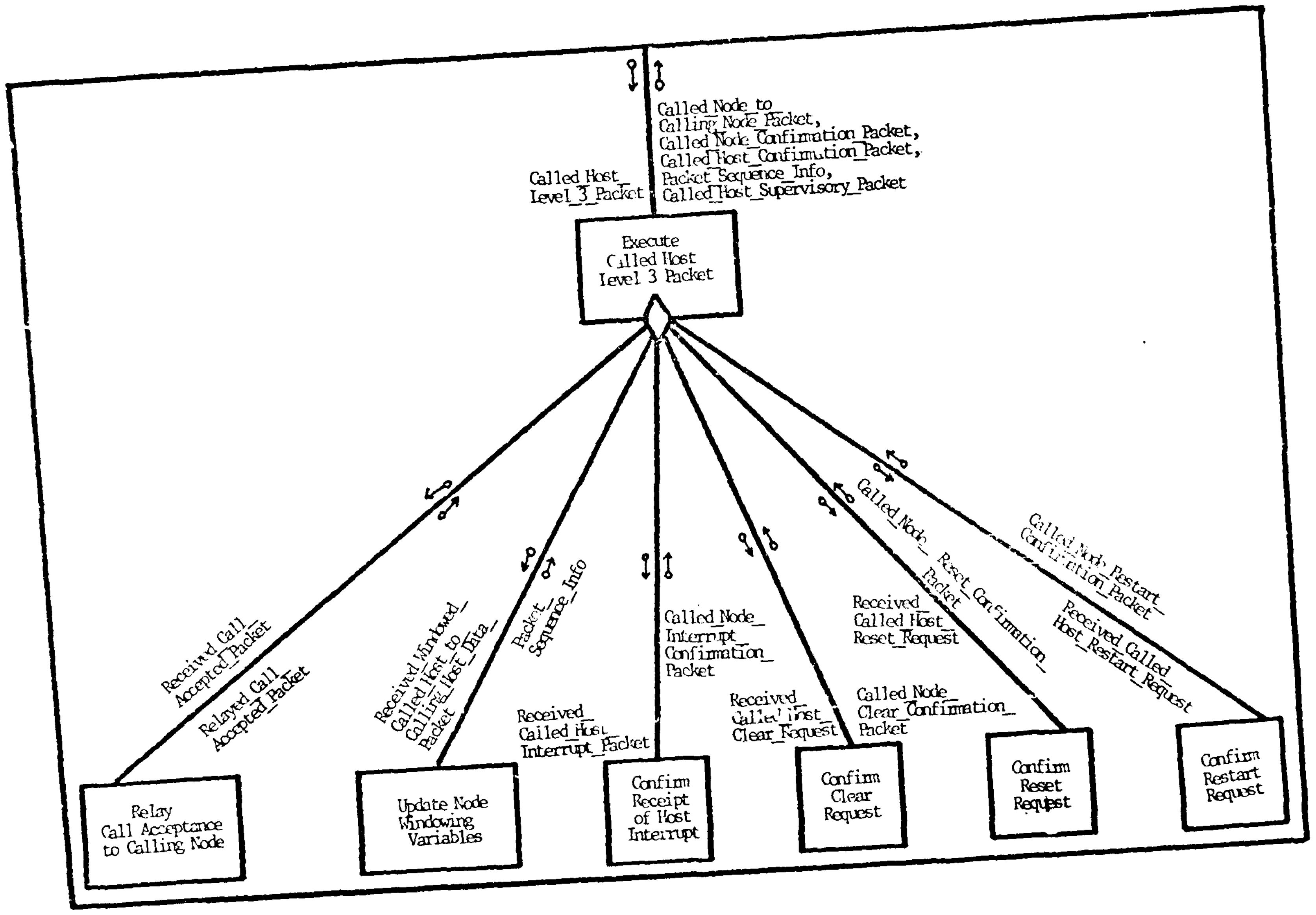

Execute Calling Node Protocol
Calling_Host_Level_3_Packet
Routed_Called_Node_Packet
Routed Called_Node_Packet, Calling_Host_Supervisory_Packet, Packet_Seq_Info
Calling_Node_Confirmation_Packet, Called_Host_to_Calling_Host_Confirmation_Packet, Error_Recovery_Packet
Calling_Node_to Called_Node_Packet
Get Host Level 3 Packet
Get Routed Node Packet
Calling_Host_Level_3_Packet
Calling_Node_to Called_Node_Packet, Calling_Node_Confirmation_Packet, Calling_Host_Confirmation_Packet, Packet_Sequence_Info, Calling_Host_Supervisory_Packet
Called_Host_to Calling_Host_Packet
to Host
Execute Calling Host Level 3 Packet
Execute Routed Called Node Packet

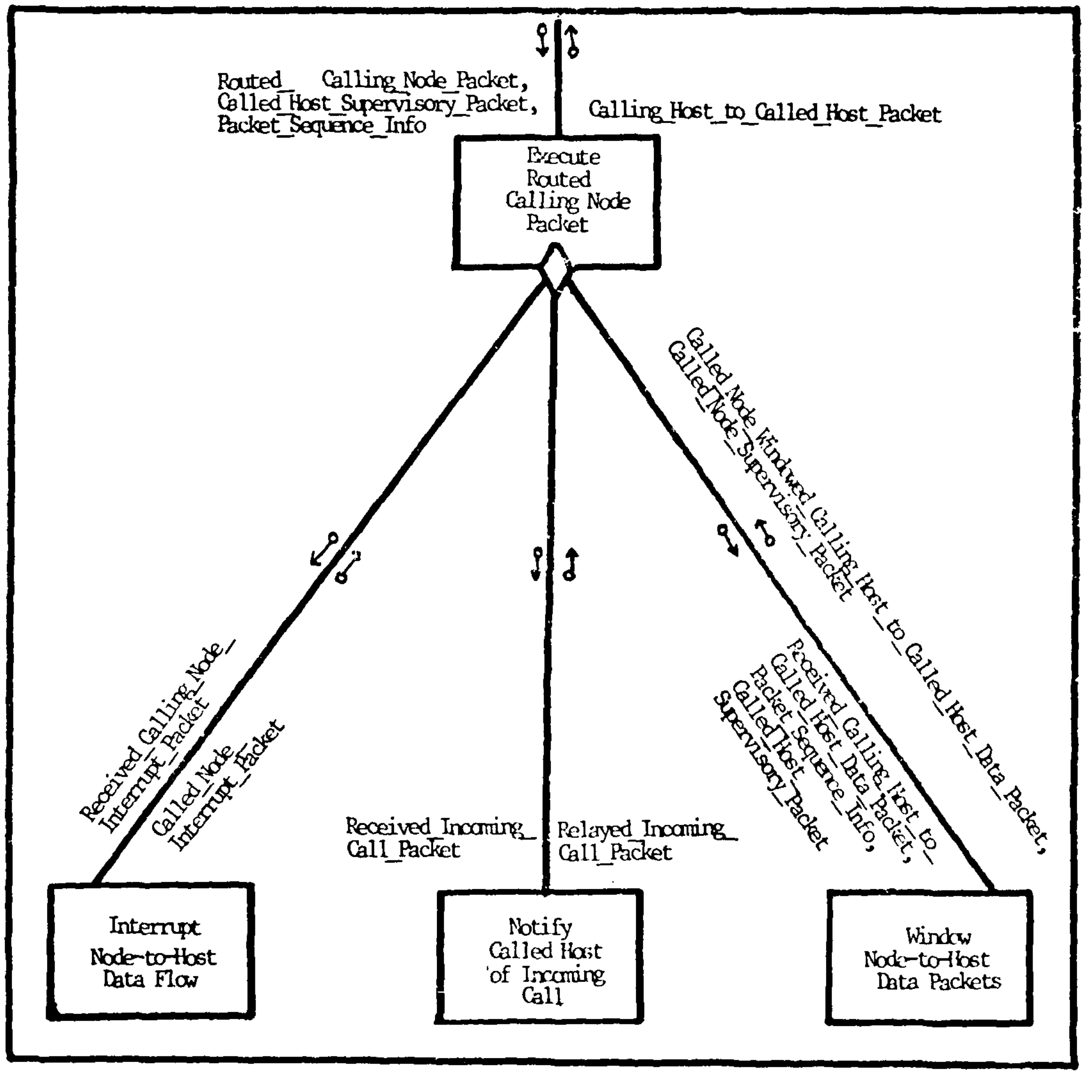


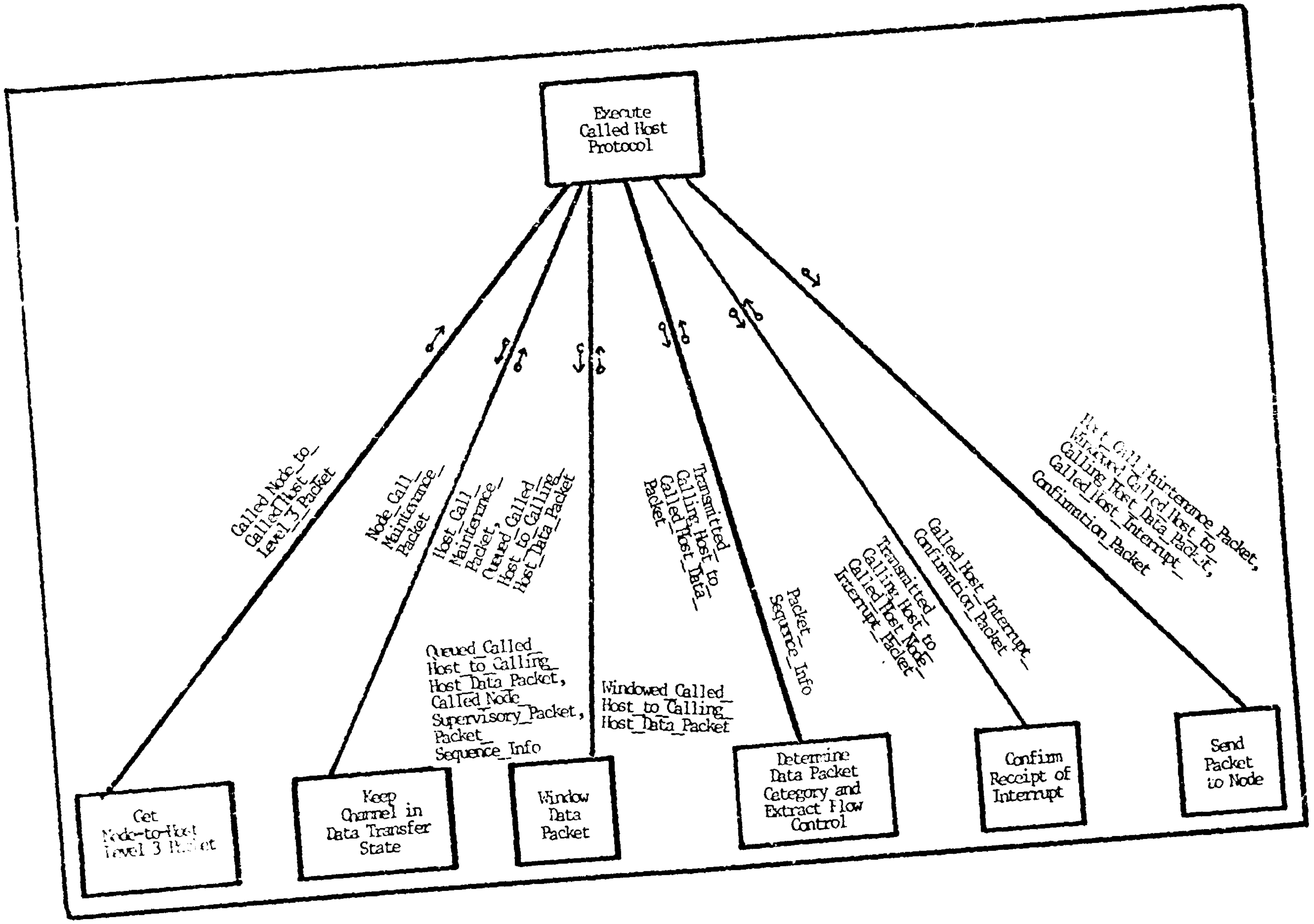
Calling_Host_Level_3_Packet
Calling_Node_to_Called_Node_Packet, Calling_Node_Confirmation_Packet, Calling_Host_Confirmation_Packet, Packet_Sequence_Info, Calling_Host_Supervisory_Packet
Execute Calling Host Level 3 Packet
Received_Call_Request_Packet
Incoming_Call_Packet
Notify Called Node of Incoming Call
Received_Windowed_Calling_Host_to_Called_Host_Data_Packet
Packet_Sequence_Info
Update Node Windowing Variables
Received_Calling_Host_Interrupt_Packet
Calling_Node_Interrupt_Confirmation_Packet
Confirm Receipt of Host Interrupt
Received_Calling_Host_Clear_Request
Calling_Node_Clear_Confirmation_Packet
Confirm Clear Request
Received_Calling_Host_Reset_Request
Calling_Node_Reset_Confirmation_Packet
Confirm Reset Request
Received_Calling_Host_Restart_Request
Calling_Node_Restart_Confirmation_Packet
Confirm Restart Request


Routed_Called_Node_Packet,
Calling_Host_Supervisory_Packet,
Packet_Sequence_Info
Called_Host_to_Calling_Host_Packet
Execute Routed Called Node Packet
Calling_Node_Windowed_Called_Host_to_Calling_Host_Data_Packet,
Calling_Node_Supervisory_Packet
Received_Called_Node_Interrupt_Packet
Calling_Node_Interrupt_Packet
Received_Called_Host_to_Calling_Host_Data_Packet,
Calling_Host_Supervisory_Packet,
Packet_Sequence_Info
Received_Call_Accepted_Packet
Call_Connected_Packet
Interrupt Node-to-Host Data Flow
Notify Calling Host of Call Connection
Window Node-to-Host Data Packets

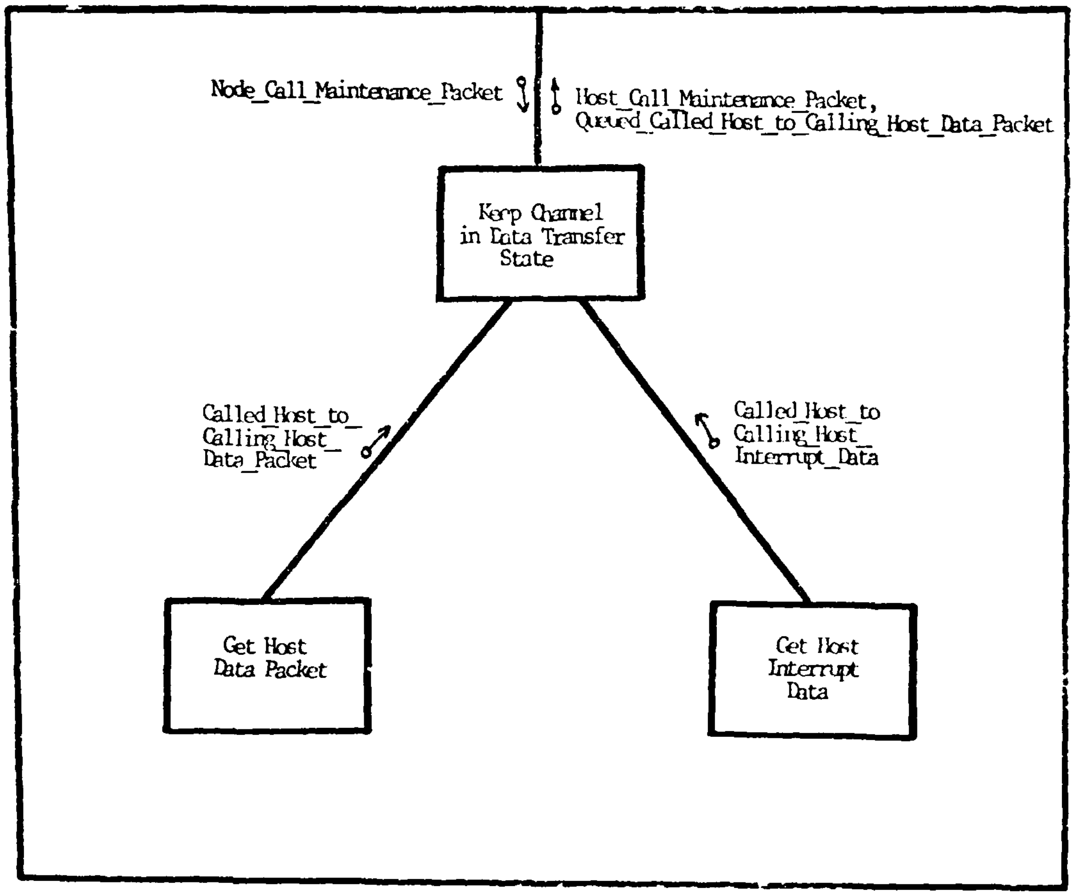

Execute Called Node Protocol
Called Host_ Level_3_Packet
Routed Calling_ Node_Packet
Called Node_to_ Calling_Node_ Packet, Called Node_ Confirmation_ Packet, Called Host_ Confirmation_ Packet, Packet_ Sequence_Info, Called Host_ Supervisory_ Packet
Routed Calling_ Node_Packet, Called Host_ Supervisory_ Packet, Packet_ Sequence_Info
Called Node_ Confirmation_Packet, Calling_Host_to_Called_ Host_Data_Packet, Error_Recovery_ Packet
Called Node_to_ Calling_Node_Packet
Get Host Level 3 Packet
Get Routed Node Packet
Called Host_ Level_3_Packet
Calling_Host_ to_Called_Host_Packet
Send Packet to Host
Send Packet on Network
Execute Called Host Level 3 Packet
Execute Routed Calling Node Packet

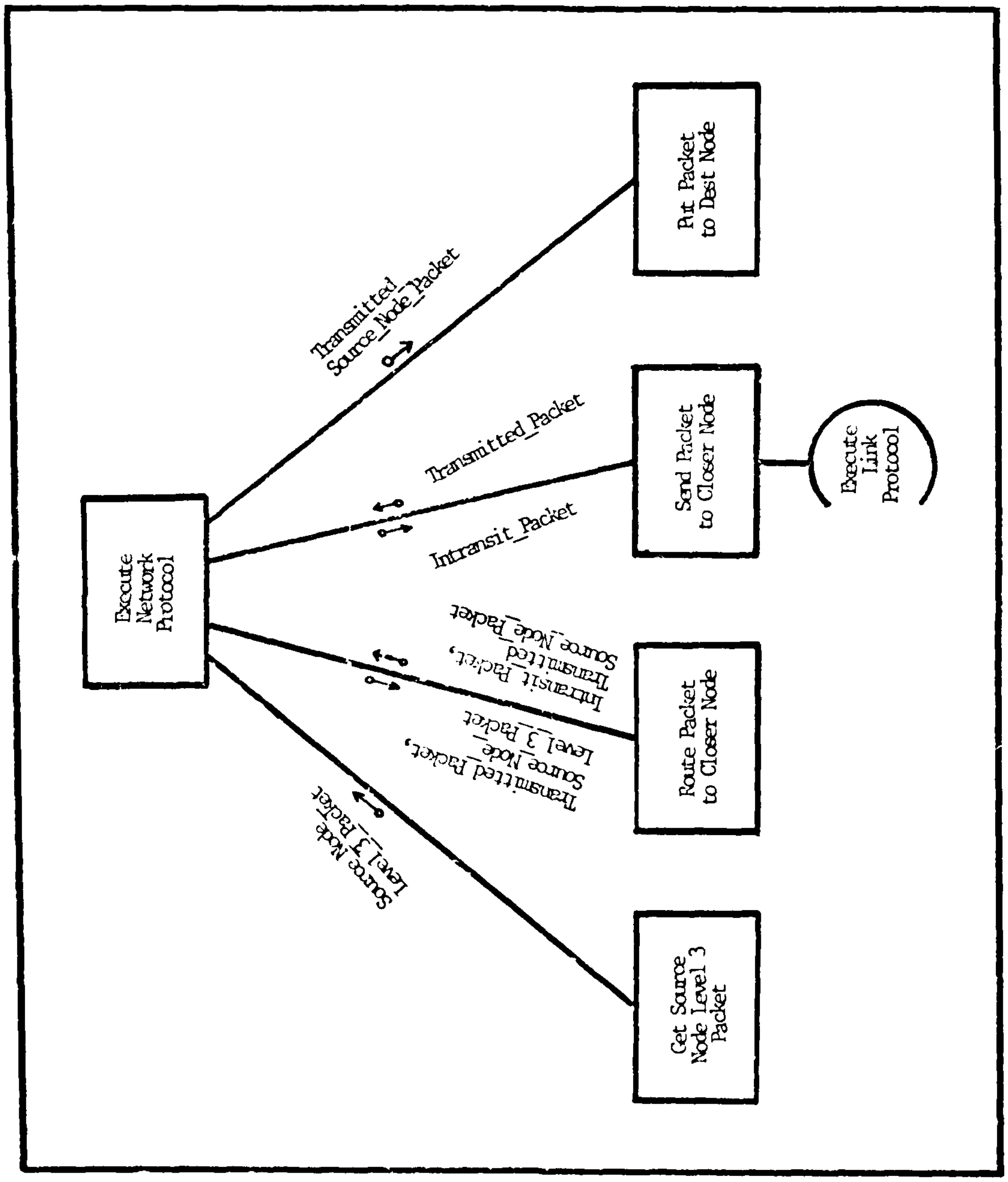

Called Host_ Level_3_Packet
Called Node_to_ Calling_Node_Packet, Called_Node_Confirmation_Packet, Called_Host_Confirmation_Packet, Packet_Sequence_Info, Called_Host_Supervisory_Packet
Execute Called Host Level 3 Packet
Received Call_ Accepted_Packet
Relayed Call_ Accepted_Packet
Received_Windowed_ Called_Host_to_ Calling_Host_Data_ Packet
Packet_ Sequence_Info
Received_ Called_Host_ Interrupt_Packet
Called_Node_ Interrupt_ Confirmation_ Packet
Received_ Called_Host_ Clear_Request
Called_Node_ Clear_Confirmation_ Packet
Received_ Called_Host_ Reset_Request
Called_Node_ Reset_Confirmation_ Packet
Received Called_ Host_Restart_Request
Called_Node_Restart_ Confirmation_Packet
Relay Call Acceptance to Calling Node
Update Node Windowing Variables
Confirm Receipt of Host Interrupt
Confirm Clear Request
Confirm Reset Request
Confirm Restart Request

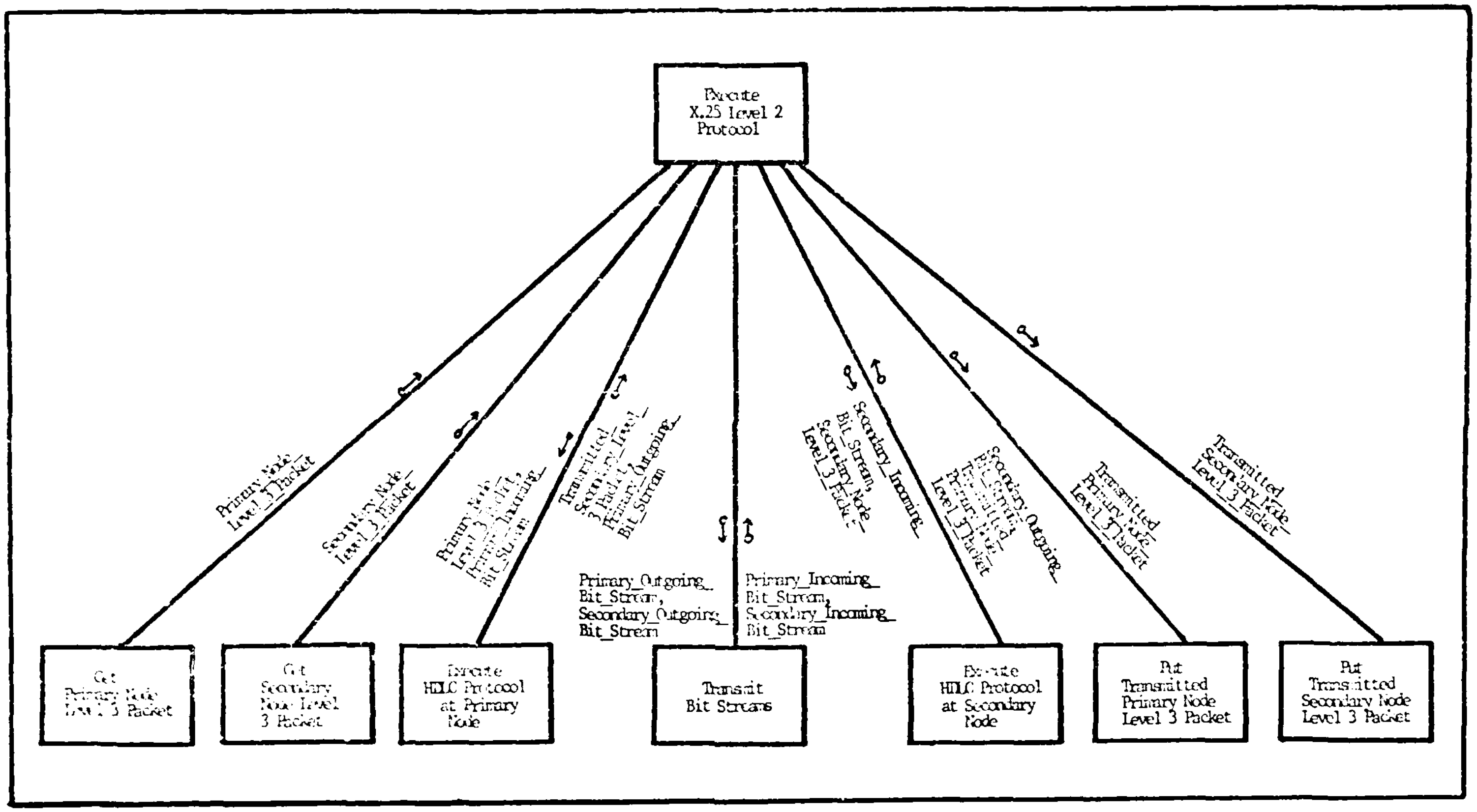

Routed_ Calling_Node_Packet,
Called_Host_Supervisory_Packet,
Packet_Sequence_Info
Calling_Host_to_Called_Host_Packet
Execute Routed Calling Node Packet
Received_Calling_Node_Interrupt_Packet
Called_Node_Interrupt_Packet
Received_Incoming_Call_Packet
Relayed_Incoming_Call_Packet
Called_Node_Windowed_Calling_Host_to_Called_Host_Data_Packet,
Called_Node_Supervisory_Packet
Received_Calling_Host_to_Called_Host_Data_Packet,
Packet_Sequence_Info,
Called_Host_Supervisory_Packet
Interrupt Node-to-Host Data Flow
Notify Called Host of Incoming Call
Window Node-to-Host Data Packets

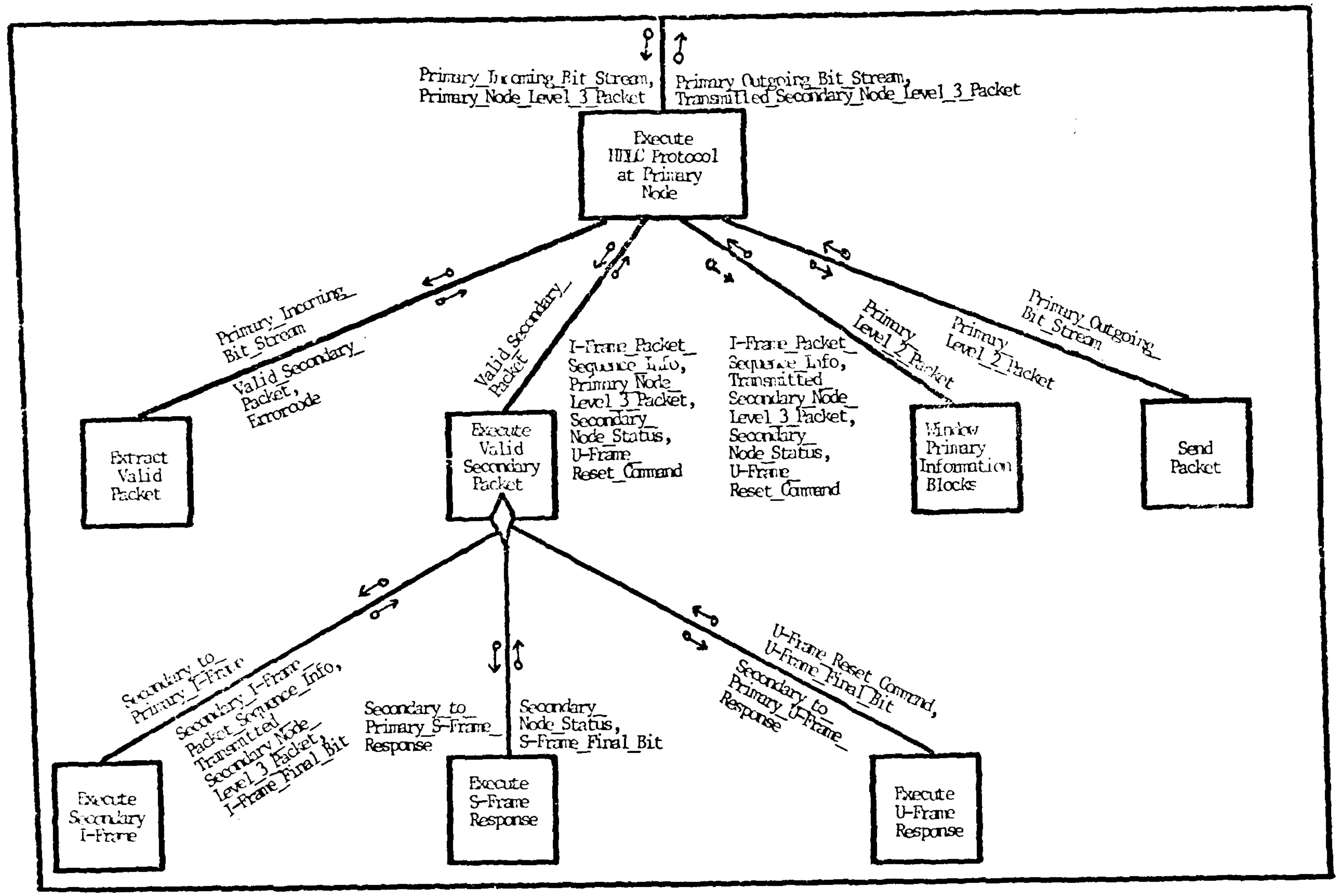


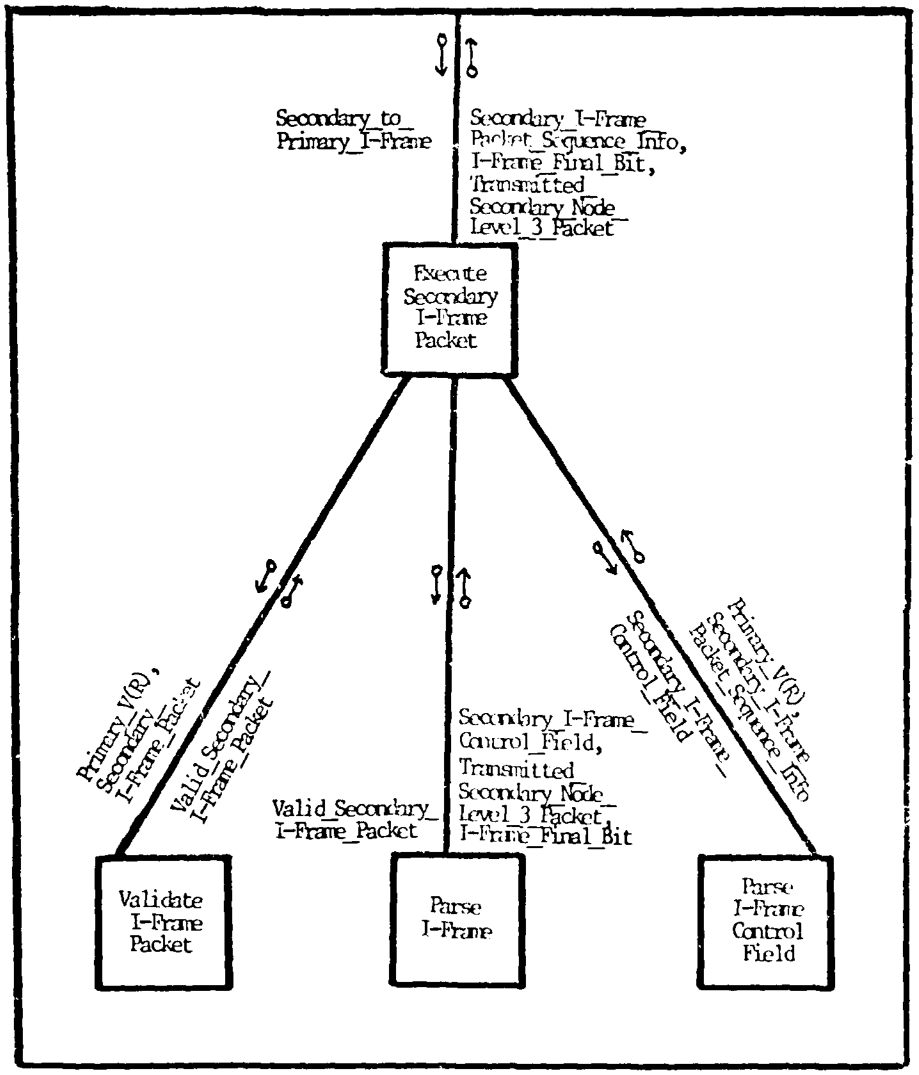
Execute Called Host Protocol
Called_Node_to_Called_Host_Level_3_Packet
Node_Call_Maintenance_Packet
Host_Call_Maintenance_Packet, Queued_Called_Host_to_Calling_Host_Data_Packet
Queued_Called_Host_to_Calling_Host_Data_Packet, Called_Node_Supervisory_Packet, Packet_Sequence_Info
Transmitted_Calling_Host_to_Called_Host_Data_Packet
Windowed_Called_Host_to_Calling_Host_Data_Packet
Packet_Sequence_Info
Transmitted_Calling_Host_to_Called_Host_Node_Interrupt_Packet
Called_Host_Interrupt_Confirmation_Packet
Called_Host_Interrupt_Confirmation_Packet
Get Node-to-Host Level 3 Packet
Keep Channel in Data Transfer State
Window Data Packet
Determine Data Packet Category and Extract Flow Control
Confirm Receipt of Interrupt
Send Packet to Node


Node_Call_Maintenance_Packet
Host_Call_Maintenance_Packet,
Queued_Called_Host_to_Calling_Host_Data_Packet
Keep Channel in Data Transfer State
Called_Host_to_Calling_Host_Data_Packet
Called_Host_to_Calling_Host_Interrupt_Data
Get Host Data Packet
Get Host Interrupt Data

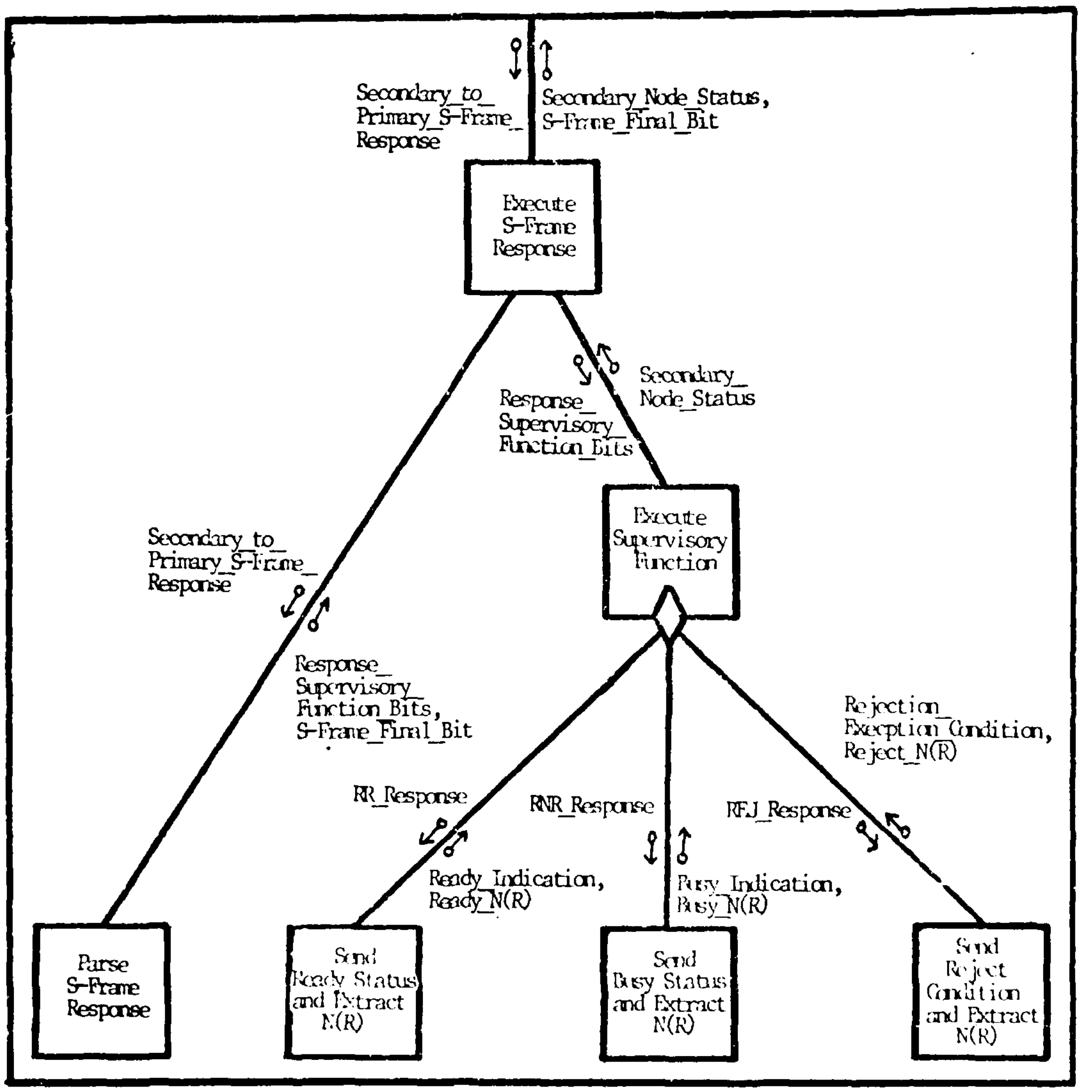

Execute Network Protocol
Put Packet to Dest Node
Send Packet to Closer Node
Route Packet to Closer Node
Get Source Node Level 3 Packet
Execute Link Protocol
Transmitted_Source_Node_Packet
Transmitted_Packet
Intransit_Packet
Intransit_Packet, Transmitted Source_Node_Packet
Transmitted_Packet, Source_Node_Level_3_Packet
Source_Node_Level_3_Packet


Execute X.25 Level 2 Protocol
Get Primary Node Level 3 Packet
Get Secondary Node Level 3 Packet
Execute HDLC Protocol at Primary Node
Transmit Bit Streams
Execute HDLC Protocol at Secondary Node
Put Transmitted Primary Node Level 3 Packet
Put Transmitted Secondary Node Level 3 Packet
Primary_Node_Level_3_Packet
Secondary_Node_Level_3_Packet
Primary_Node_Level_3_Packet, Primary_Incoming_Bit_Stream
Transmitted_Secondary_Level_3_Packet, Primary_Outgoing_Bit_Stream
Primary_Outgoing_Bit_Stream, Secondary_Outgoing_Bit_Stream
Primary_Incoming_Bit_Stream, Secondary_Incoming_Bit_Stream
Secondary_Incoming_Bit_Stream, Secondary_Node_Level_3_Packet
Secondary_Outgoing_Bit_Stream, Transmitted_Primary_Node_Level_3_Packet
Transmitted_Primary_Node_Level_3_Packet
Transmitted_Secondary_Node_Level_3_Packet

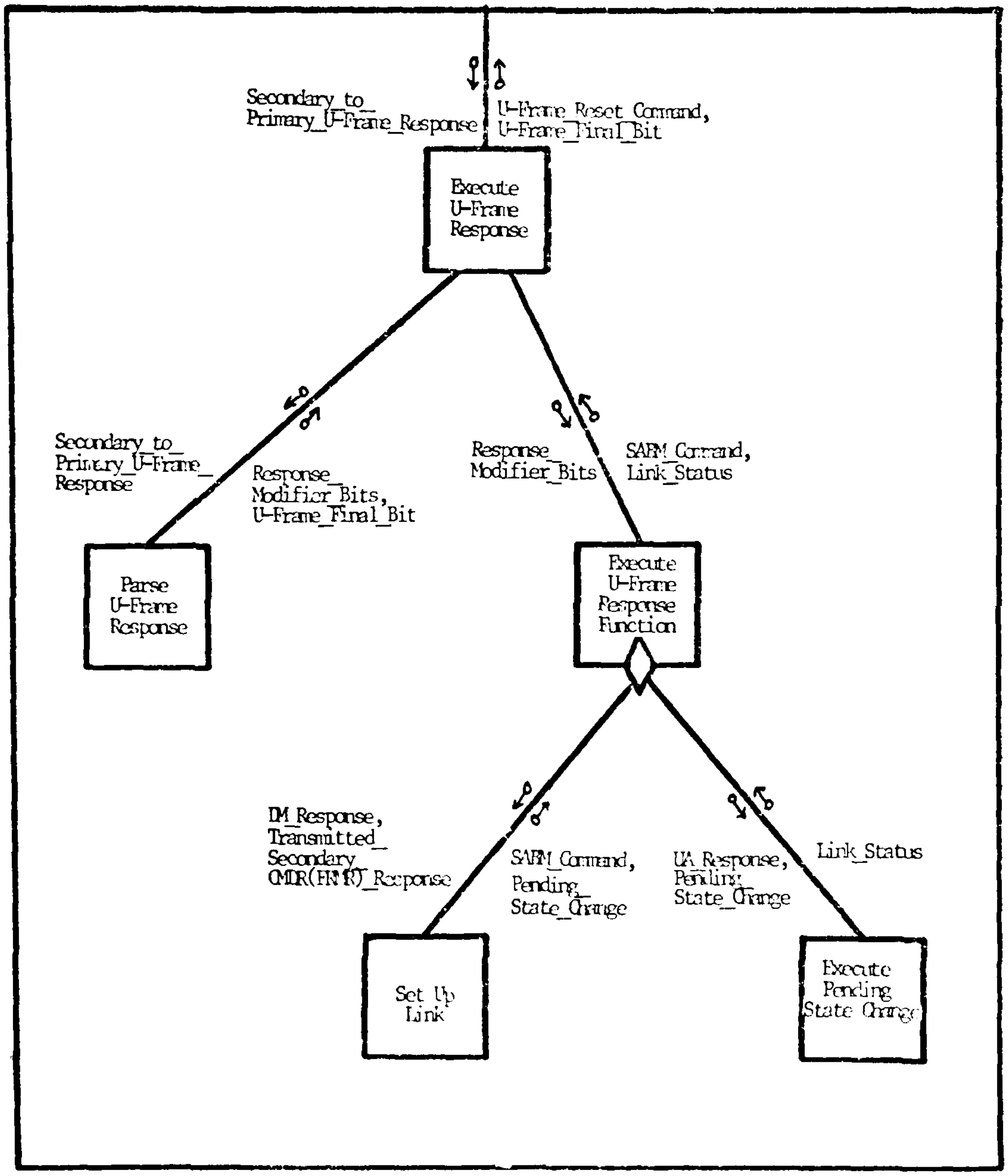

Primary_Incoming_Bit_Stream,
Primary_Node_Level_3_Packet
Primary_Outgoing_Bit_Stream,
Transmitted_Secondary_Node_Level_3_Packet
Execute HDLC Protocol at Primary Node
Primary_Incoming_Bit_Stream
Valid_Secondary_Packet, Errorcode
Valid_Secondary_Packet
I-Frame_Packet_Sequence_Info, Primary_Node_Level_3_Packet, Secondary_Node_Status, U-Frame_Reset_Command
I-Frame_Packet_Sequence_Info, Transmitted_Secondary_Node_Level_3_Packet, Secondary_Node_Status, U-Frame_Reset_Command
Primary_Level_2_Packet
Primary_Level_2_Packet
Primary_Outgoing_Bit_Stream
Extract Valid Packet
Execute Valid Secondary Packet
Window Primary Information Blocks
Send Packet
Secondary_to_Primary_I-Frame
Secondary_I-Frame_Packet_Sequence_Info, Transmitted_Secondary_Node_Level_3_Packet, I-Frame_Final_Bit
Secondary_to_Primary_S-Frame_Response
Secondary_Node_Status, S-Frame_Final_Bit
U-Frame_Reset_Command, U-Frame_Final_Bit
Secondary_to_Primary_U-Frame_Response
Execute Secondary I-Frame
Execute S-Frame Response
Execute U-Frame Response

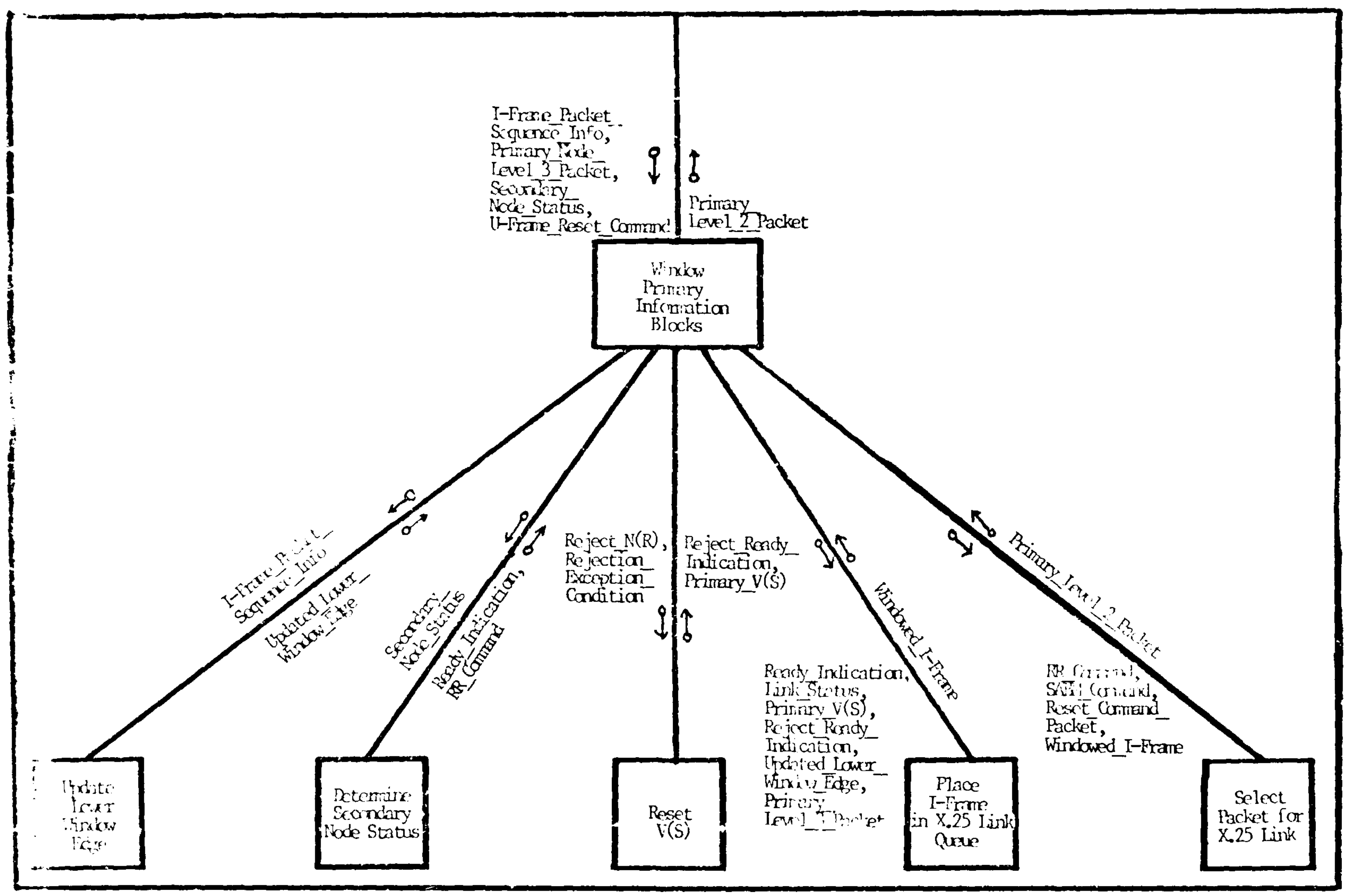


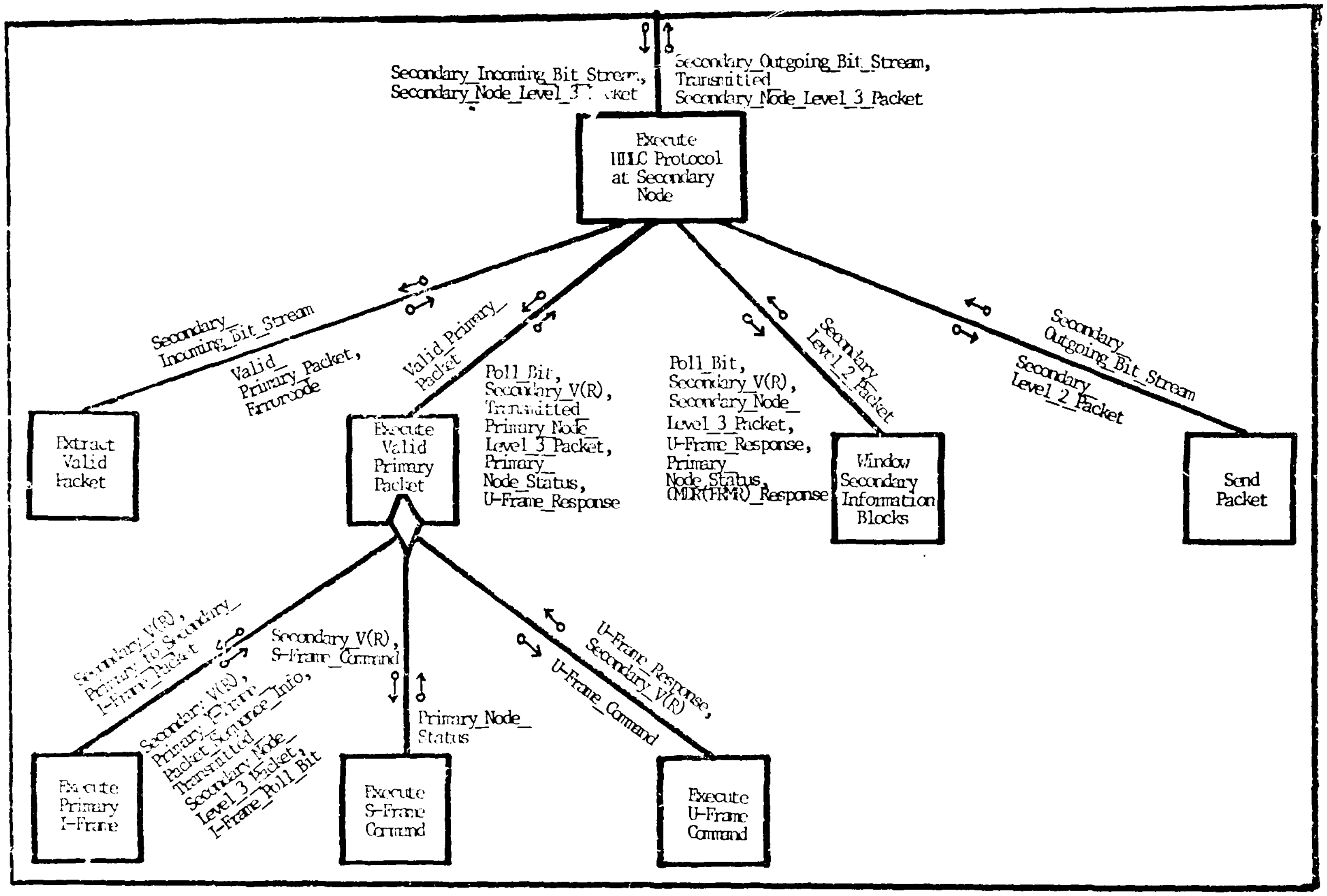
Secondary_to_
Primary_I-Frame
Secondary_I-Frame_
Packet_Sequence_Info,
I-Frame_Final_Bit,
Transmitted_
Secondary_Node_
Level_3_Packet
Execute
Secondary
I-Frame
Packet
Primary_V(R),
Secondary_
I-Frame_Packet
Valid_Secondary_
I-Frame_Packet
Valid_Secondary_
I-Frame_Packet
Secondary_I-Frame_
Control_Field,
Transmitted_
Secondary_Node_
Level_3_Packet,
I-Frame_Final_Bit
Secondary_I-Frame_
Control_Field
Primary_V(R),
Secondary_I-Frame_
Packet_Sequence_Info
Validate
I-Frame
Packet
Parse
I-Frame
Parse
I-Frame
Control
Field



Secondary_to_Primary_S-Frame_Response
Secondary_Node_Status, S-Frame_Final_Bit
Execute S-Frame Response
Response_Supervisory_Function_Bits
Secondary_Node_Status
Execute Supervisory Function
Secondary_to_Primary_S-Frame_Response
Response_Supervisory_Function_Bits, S-Frame_Final_Bit
Rejection_Exception_Condition, Reject_N(R)
RR_Response
RNR_Response
REJ_Response
Ready_Indication, Ready_N(R)
Busy_Indication, Busy_N(R)
Parse S-Frame Response
Send Ready Status and Extract N(R)
Send Busy Status and Extract N(R)
Send Reject Condition and Extract N(R)


Secondary_to_Primary_U-Frame_Response
U-Frame_Reset_Command, U-Frame_Final_Bit
Execute U-Frame Response
Secondary_to_Primary_U-Frame_Response
Response_Modifier_Bits, U-Frame_Final_Bit
Response_Modifier_Bits
SARM_Command, Link_Status
Parse U-Frame Response
Execute U-Frame Response Function
DM_Response, Transmitted_Secondary_CMDR(FRMR)_Response
SARM_Command, Pending_State_Change
UA_Response, Pending_State_Change
Link_Status
Set Up Link
Execute Pending State Change


I-Frame_Packet_Sequence_Info, Primary_Node_Level_3_Packet, Secondary_Node_Status, U-Frame_Reset_Command
Primary_Level_2_Packet
Window Primary Information Blocks
I-Frame_Packet_Sequence_Info
Updated_Lower_Window_Edge
Secondary_Node_Status
Ready_Indication, RR_Command
Reject_N(R), Rejection_Exception_Condition
Reject_Ready_Indication, Primary_V(S)
Ready_Indication, Link_Status, Primary_V(S), Reject_Ready_Indication, Updated_Lower_Window_Edge, Primary_Level_2_Packet
Windowed_I-Frame
Primary_Level_2_Packet
RR_Command, SABM_Command, Reset_Command_Packet, Windowed_I-Frame
Update Lower Window Edge
Determine Secondary Node Status
Reset V(S)
Place I-Frame in X.25 Link Queue
Select Packet for X.25 Link

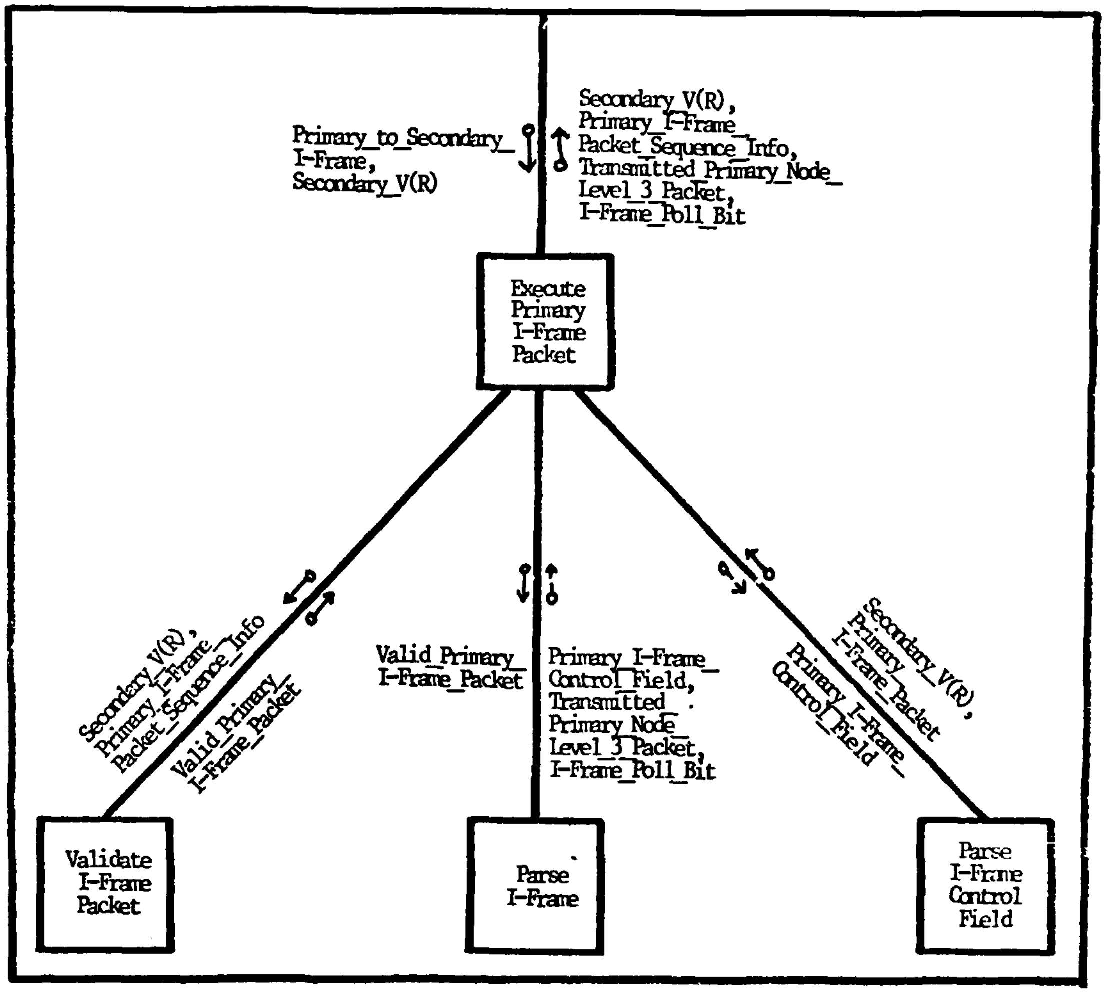


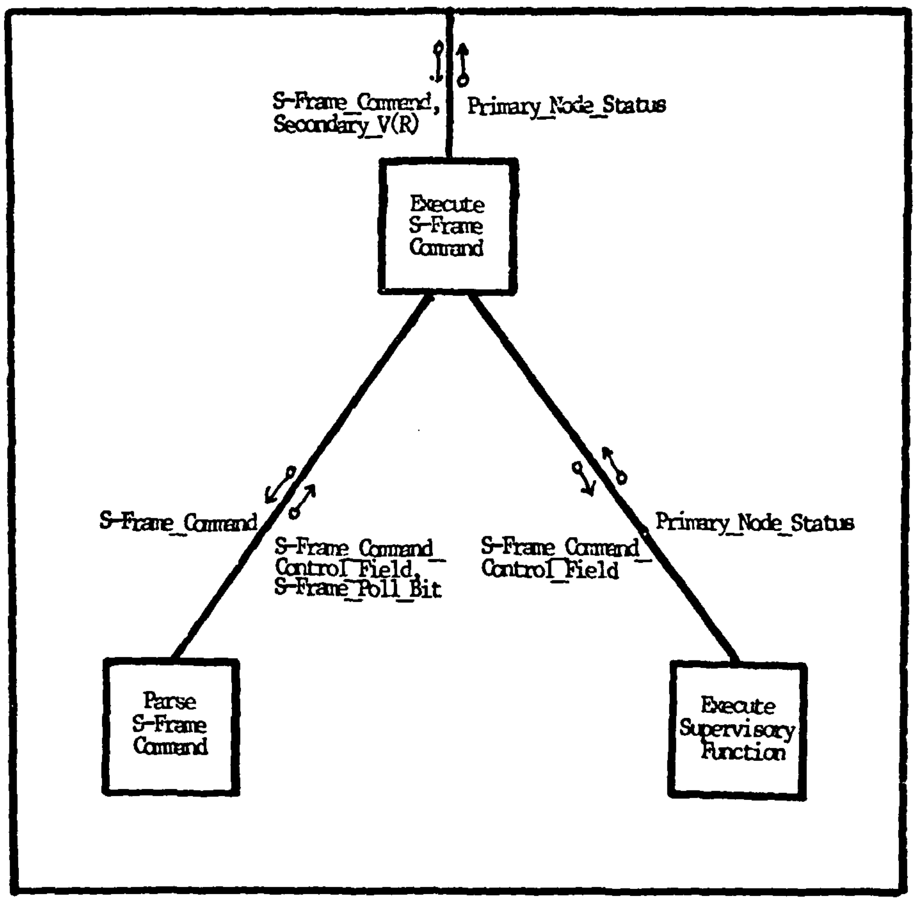
Secondary_Incoming_Bit_Stream,
Secondary_Node_Level_3_Packet
Secondary_Outgoing_Bit_Stream,
Transmitted
Secondary_Node_Level_3_Packet
Execute HDLC Protocol at Secondary Node
Secondary_Incoming_Bit_Stream
Valid_Primary_Packet,
Errorcode
Valid_Primary_Packet
Poll_Bit,
Secondary_V(R),
Transmitted
Primary_Node_
Level_3_Packet,
Primary_
Node_Status,
U-Frame_Response
Poll_Bit,
Secondary_V(R),
Secondary_Node_
Level_3_Packet,
U-Frame_Response,
Primary_
Node_Status,
CMDR(FRMR)_Response
Secondary_Level_2_Packet
Secondary_Outgoing_Bit_Stream
Secondary_Level_2_Packet
Extract Valid Packet
Execute Valid Primary Packet
Window Secondary Information Blocks
Send Packet
Secondary_V(R),
Primary_to_Secondary_
I-Frame_Packet
Secondary_V(R),
Primary_I-Frame_
Packet_Sequence_Info,
Transmitted_
Secondary_Node_
Level_3_Packet,
I-Frame_Poll_Bit
Secondary_V(R),
S-Frame_Command
Primary_Node_
Status
U-Frame_Response,
Secondary_V(R)
U-Frame_Command
Execute Primary I-Frame
Execute S-Frame Command
Execute U-Frame Command

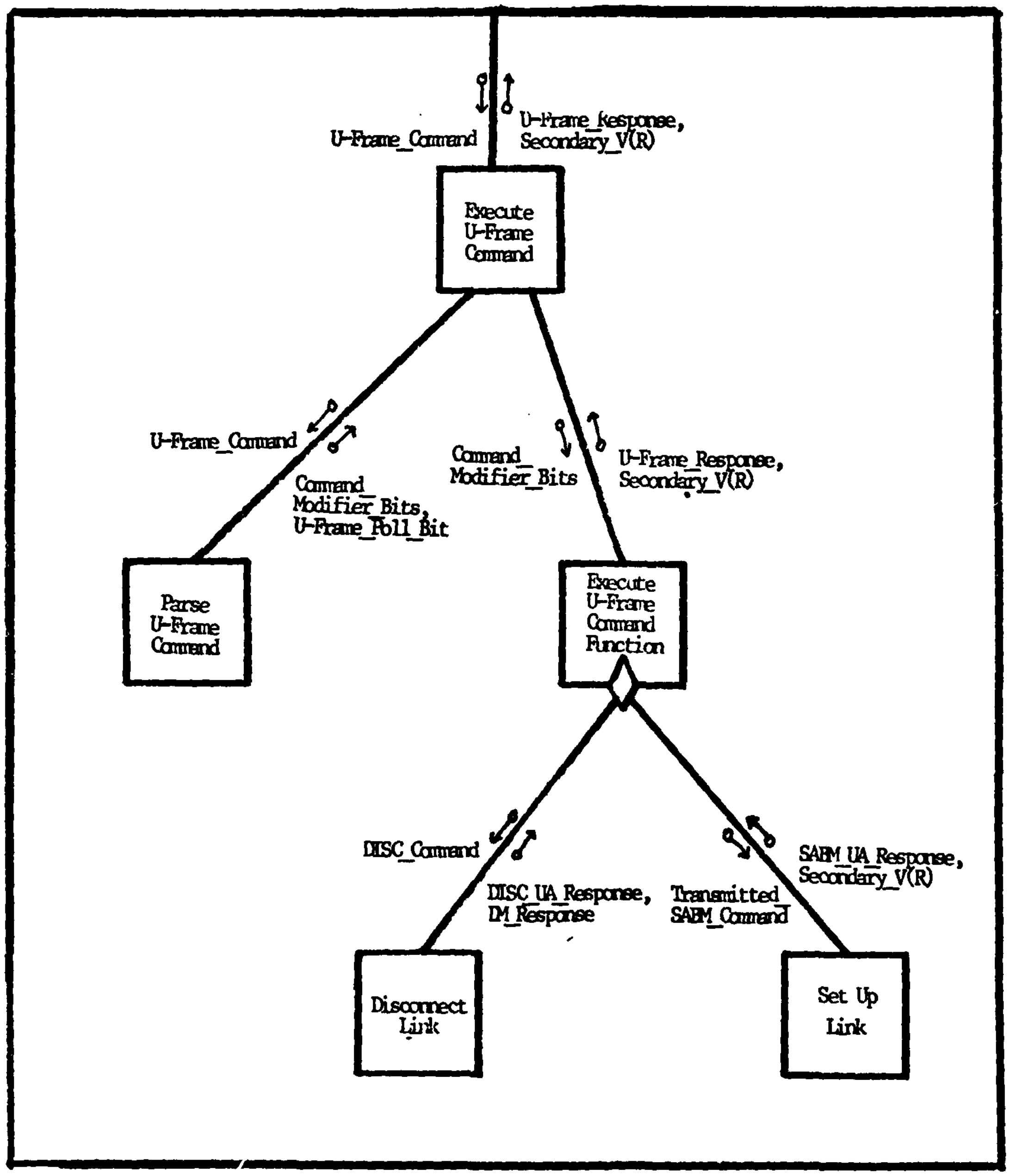
AD-A100 822 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/6 9/2
DESIGN OF A LOCAL COMPUTER NETWORK FOR THE AIR FORCE INSTITUTE --ETC(U)
MAR 81 W C HOBART
UNCLASSIFIED AFIT/GE/EE/81M-3 NL

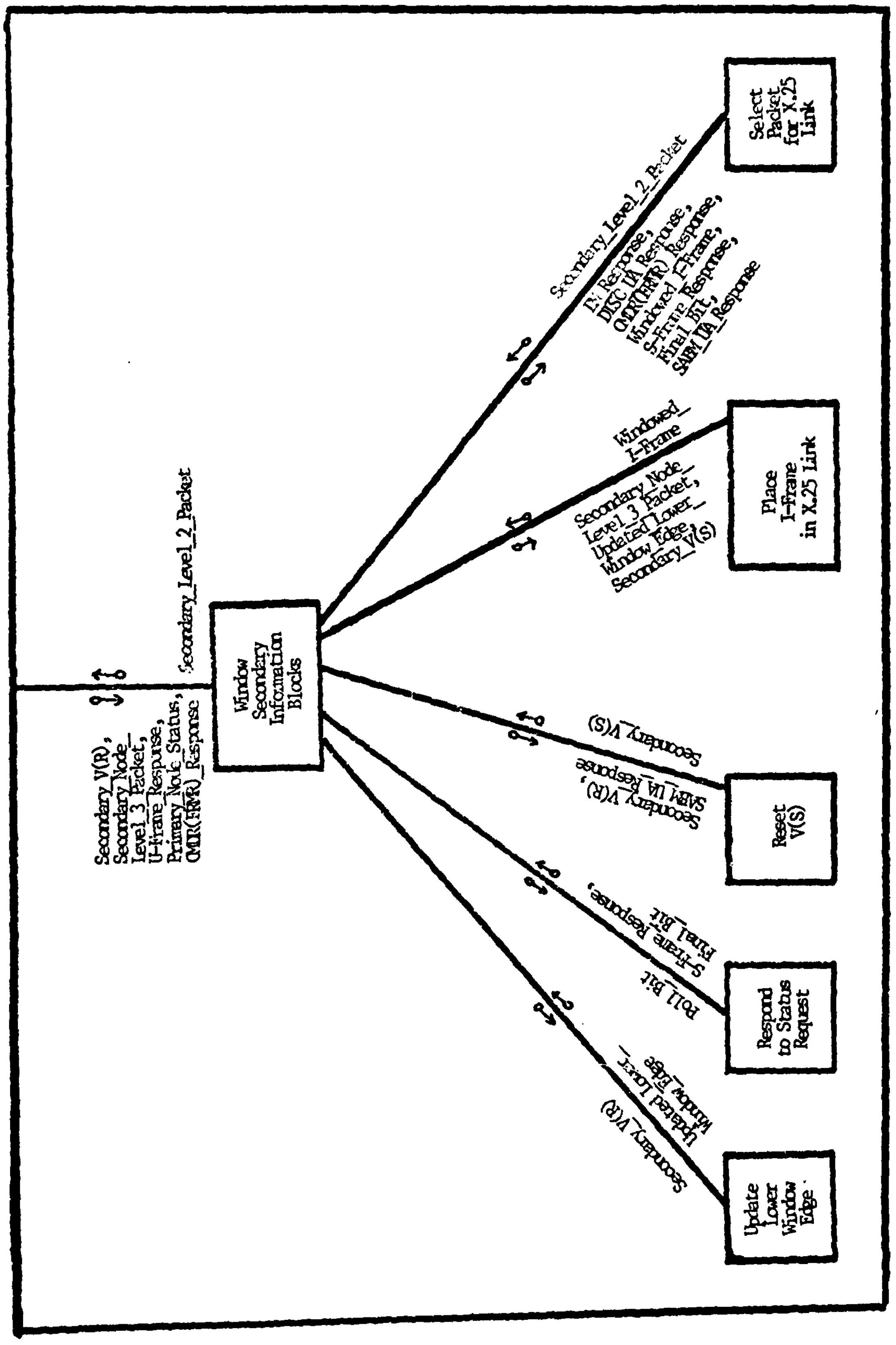
4 OF 4
AD A
100822

END
DATE
FILMED
7-81
DTIC


Primary_to_Secondary_
I-Frame,
Secondary_V(R)
Secondary_V(R),
Primary_I-Frame_
Packet_Sequence_Info,
Transmitted_Primary_Node_
Level_3_Packet,
I-Frame_Poll_Bit
Execute Primary I-Frame Packet
Secondary_V(R),
Primary_I-Frame_
Packet_Sequence_Info
Valid_Primary_
I-Frame_Packet
Valid_Primary_
I-Frame_Packet
Primary_I-Frame_
Control_Field,
Transmitted_
Primary_Node_
Level_3_Packet,
I-Frame_Poll_Bit
Secondary_V(R),
Primary_
I-Frame_Packet
Primary_I-Frame_
Control_Field
Validate I-Frame Packet
Parse I-Frame
Parse I-Frame Control Field


S-Frame_Command,
Secondary_V(R)
Primary_Node_Status
Execute S-Frame Command
S-Frame_Command
S-Frame_Command_Control_Field,
S-Frame_Poll_Bit
S-Frame_Command_Control_Field
Primary_Node_Status
Parse S-Frame Command
Execute Supervisory Function


U-Frame_Command
U-Frame_Response, Secondary_V(R)
Execute U-Frame Command
U-Frame_Command
Command_Modifier_Bits, U-Frame_Poll_Bit
Command_Modifier_Bits
U-Frame_Response, Secondary_V(R)
Parse U-Frame Command
Execute U-Frame Command Function
DISC_Command
DISC_UA_Response, DM_Response
Transmitted_SABM_Command
SABM_UA_Response, Secondary_V(R)
Disconnect Link
Set Up Link


Window Secondary Information Blocks
Select Packet for X.25 Link
Place I-Frame in X.25 Link
Reset V(S)
Respond to Status Request
Update Lower Window Edge
Secondary_Level_2_Packet
Secondary_V(R), Secondary_Node_Level_3_Packet, U-Frame_Response, Primary_Node_Status, CMDR(FRMR)_Response
Secondary_Level_2_Packet
RR_Response, DISC_UA_Response, CMDR(FRMR)_Response, Windowed_I-Frame, S-Frame_Response, Final_Bit, SABM_UA_Response
Windowed_I-Frame
Secondary_Node_Level_3_Packet, Updated_Lower_Window_Edge, Secondary_V(S)
Secondary_V(S)
Secondary_V(R), SABM_UA_Response
S-Frame_Response, Final_Bit
Poll_Bit
Secondary_V(R)
Updated_Lower_Window_Edge

Appendix E

Source Code for the Network Command Language Interpreter

This appendix contains the PASCAL source code for the network command language interpreter. In its present configuration, the program is structured to allow testing of the high-level protocol modules.

```
TESTEXTRNETCMD   4-DEC-1980   10:17:35   VAX-11 PASCAL VERSION V1.0-1        PAGE   1
01               4-DEC-1980   10:14:07   DMA1:[HOBART]EXTRNET.PAS;44 (1)

      LINE       LEVEL
     NUMBERS   PROC STMT STATEMENT.        .        .        .        .        .

       100      1    1       PROGRAM TESTEXTRNETCMD(INPUT,OUTPUT);
       200      2    1
       300      3    1
       400      4    1         CONST
       500      5    1           MAXCMDLNG=120;
       600      6    1           ENDLN='!';
       700      7    1
       800      8    1
       900      9    1         TYPE
      1000     10    1           STRGTYPE=PACKED ARRAY [1..30] OF CHAR;
      1100     11    1           CMDTYPE=ARRAY [1..MAXCMDLNG] OF CHAR;
      1200     12    1
      1300     13    1
      1400     14    1         VAR
      1500     15    1           I,INDEX,NUMTESTCASES: INTEGER;
      1600     16    1           NETRESP: TEXT;
      1700     17    1           SOURCE: CMDTYPE;
      1800     18    1
      1900     19    1
      2000     20    2       PROCEDURE EXTRNETCMD(TRNETCMD: CMDTYPE;VAR NETRESP: TEXT);
      2100     21    2
      2200     22    2         CONST
      2300     23    2           MAXPARLNG=20;
      2400     24    2           ENDLN='!';
      2500     25    2           FILLER='#';
      2600     26    2           MAXERRMSG=40;
      2700     27    2           NONE='########################################';
      2800     28    2
      2900     29    2
      3000     30    2         TYPE
      3100     31    2           PARTYPE=ARRAY [1..MAXPARLNG] OF CHAR;
      3200     32    2           ERRORTYPE=PACKED ARRAY [1..MAXERRMSG] OF CHAR;
      3300     33    2           SETTYPE=SET OF CHAR;
      3400     34    2
      3500     35    2
      3600     36    2         VAR
      3700     37    2           VALIDCMD: BOOLEAN;
      3800     38    2           I: INTEGER;
      3900     39    2           CMDFIELD: PARTYPE;
      4000     40    2           PARAMS: CMDTYPE;
      4100     41    2           COMMANDSET: SET OF CHAR;
      4200     42    2           ERRORMSG: ERRORTYPE;
      4300     43    2           LISTACTDEV,LISTACTHOST: SETTYPE;
      4400     44    2
      4500     45    2
      4600     46    2
```

```
5200   TESTEXTRNETCMD   4-DEC-1980   10:17:35   VAX-11 PASCAL VERSION V1.0-1        PAGE   2
5300   01               4-DEC-1980   10:14:07   DMA1:[HOBART]EXTRNET.PAS;44 (2)

5400        LINE       LEVEL
5500       NUMBERS   PROC STMT STATEMENT.        .        .        .        .            .
5600
5700              47   2
5800     4700     48   3         PROCEDURE COPY1PARAM(SPARAM: CMDTYPE;VAR DPARAM: PARTYPE;
5900     4800     49   3                                     VAR INDEX: INTEGER;VAR VALPARAM: BOOLEAN);
6000     4900     50   3
6100     5000     51   3         LABEL 900;
6200     5100     52   3
6300     5200     53   3
6400     5300     54   3         CONST
6500     5400     55   3           SEPARATOR=',';
6600     5500     56   3
6700     5600     57   3
6800     5700     58   3         TYPE
6900     5800     59   3           PARIND=1..MAXPARLNG;
7000     5900     60   3
7100     6000     61   3
7200     6100     62   3         VAR
7300     6200     63   3           INDEXP1,J: INTEGER;
7400     6300     64   3
7500     6400     65   4         FUNCTION ENDPARAM(INDEX: INTEGER;SOURCE: CMDTYPE): BOOLEAN;
7600     6500     66   4         BEGIN (*DETERMINE IF END OF PARAMETER*)
7700     6600     67        0      IF (SOURCE[INDEX]=SEPARATOR) OR (SOURCE[INDEX]=ENDLN) THEN ENDPARAM:=TRUE
7800     6700     68        1      ELSE ENDPARAM:=FALSE;
7900     6800     69        1    END; (*DETERMINE IF END OF PARAMETER*)
8000     6900     70        0
8100     7000     71        0    BEGIN (*COPY ONE PARAMETER*)
8200     7100     72        0      J:=1;
8300     7200     73        1      IF INDEX<=MAXCMDLNG THEN BEGIN
8400     7300     74        2        VALPARAM:=TRUE; (*POSIT VALID PARAMETER*)
8500     7400     75        2        WHILE NOT ENDPARAM(INDEX,SPARAM) DO BEGIN
8600     7500     76        3          DPARAM[J]:=SPARAM[INDEX];
8700     7600     77        3          INDEXP1:=INDEX+1;
8800     7700     78        3          IF (INDEXP1>MAXCMDLNG) THEN BEGIN
8900     7800     79        4            VALPARAM:=FALSE;
9000     7900     80        4            J:=1;
9100     8000     81        4            GOTO 900; END (*QUIT POSIT*)
9200     8100     82        4          ELSE IF ((J+1)>MAXPARLNG) AND NOT ENDPARAM(INDEXP1,SPARAM) THEN BEGIN
9300     8200     83        4            VALPARAM:=FALSE;
9400     8300     84        4            J:=1;
9500     8400     85        4            GOTO 900; END (*QUIT POSIT*)
9600     8500     86        4          ELSE BEGIN
9700     8600     87        4              INDEX:=INDEX+1;
9800     8700     88        4              J:=J+1; END; (*IF*)
9900     8800     89        3        END; (*WHILE*)
10000    8900     90        2        IF SPARAM[I]=SEPARATOR THEN INDEX:=INDEX+1; END (*THEN*)
10100    9000     91        2      ELSE VALPARAM:=FALSE;
10200    9100     92        1    900: WHILE J<=MAXPARLNG DO BEGIN
10300    9200     93        2          DPARAM[J]:=FILLER;
10400    9300     94        2          J:=J+1; END; (*WHILE*)
10500    9400     95        1    END; (*COPY ONE PARAMETER*)
10600    9500     96        0
10700    9600     97        0
```

```
TESTEXTRNETCMD   4-DEC-1980   10:17:35   VAX-11 PASCAL VERSION V1.0-1        PAGE   3
01               4-DEC-1980   10:14:07   DMA1:[HOBART]EXTRNET.PAS;44 (3)

      LINE       LEVEL
     NUMBERS   PROC STMT STATEMENT.           .         .         .         .         .

               98         0
       9700    99    3       PROCEDURE GETPARAM(ERRORMSG: ERRORTYPE;VAR PARAM: PARTYPE);
       9800   100    3
       9900   101    3         CONST
      10000   102    3           MAXPARLNG=20;
      10100   103    3
      10200   104    3         VAR
      10300   105    3           I: INTEGER;
      10400   106    3
      10500   107    3         BEGIN (*GET PARAMETER FROM USER*)
      10600   108         0      FOR I:=1 TO MAXERRMSG DO WRITE(ERRORMSG[I]);
      10700   109         1      I:=1;
      10800   110         1      WHILE (I<=MAXPARLNG) AND (NOT EOLN) DO BEGIN
      10900   111         2        READ(PARAM[I]);
      11000   112         2        I:=I+1;  END; (*WHILE*)
      11100   113         1      READLN;
      11200   114         1      WHILE I<=MAXPARLNG DO BEGIN
      11300   115         2        PARAM[I]:=FILLER;
      11400   116         2        I:=I+1;  END; (*WHILE*)
      11500   117         1    END; (*GET PARAMETER FROM USER*)
      11600   118         0
      11700   119         0
```

```
TESTEXTRNETCMD   4-DEC-1980   10:17:35   VAX-11 PASCAL VERSION V1.0-1        PAGE   4
01               4-DEC-1980   10:14:07   DMA1:[HOBART]EXTRNET.PAS;44 (4)

    LINE        LEVEL
   NUMBERS    PROC STMT STATEMENT.          .          .          .          .                              .

           120         0
   11800   121    3         PROCEDURE HELPUSER(HELPREQ: CMDTYPE;VAR HELPINFO: TEXT);
   11900   122    3
   12000   123    3           VAR
   12100   124    3             HELPSET: SET OF CHAR;
   12110   125    3             ERROR: ERRORTYPE;
   12120   126    3             HELPPARAM: PARTYPE;
   12130   127    3             VALCOPY: BOOLEAN;
   12140   128    3             I: INTEGER;
   12500   129    3
   12600   130    3           BEGIN(*HELP USER*)
   12700   131         0        HELPSET:=['F','L','S',FILLER];
   12702   132         1        I:=1;
   12703   133         1        COPY1PARAM(HELPREQ,HELPPARAM,I,VALCOPY);
   12710   134         1        IF NOT VALCOPY OR NOT (HELPPARAM[1] IN HELPSET) THEN BEGIN
   12720   135         2          ERROR:='INVALID HELP REQUEST PARAMETER--PARAM?> ';
   12730   136         2          REPEAT GETPARAM(ERROR,HELPPARAM) UNTIL HELPPARAM[1] IN HELPSET;
   12740   137         2          END; (*IF*)
   12750   138         1        REWRITE(HELPINFO);
   12760   139         1        CASE HELPPARAM[1] OF
   12770   140         2          'F': BEGIN (*DESCRIBE FILE TRANSFER COMMAND*)
   12780   141         3                 WRITELN(HELPINFO,'FILE TRANSFER COMMAND FORMAT IS AS FOLLOWS:');
   12790   142         3                 WRITE(HELPINFO,'NETWORK,T)RANSFER FILE,FN=FILENAME,');
   12800   143         3                 WRITE(HELPINFO,'DD=DESTDEV,DH=DESTHOST,SD=SOURCEDEV,');
   12810   144         3                 WRITELN(HELPINFO,'SH=SOURCEHOST');
   12820   145         3                 WRITE(HELPINFO,'T) MEANS THAT TRANSFER FILE MAY BE');
   12825   146         3                 WRITELN(HELPINFO,' ABBREVIATED');
   12827   147         3                 WRITELN(HELPINFO,'PARAMETERS MAY BE ENTERED IN ANY ORDER');
   12830   148         3                 WRITELN(HELPINFO,'ALL PARAMETERS ARE LIMITED TO 20 CHAR');
   12840   149         3                 WRITELN(HELPINFO,'SOURCEDEV CANNOT BE THE CONSOLE OR PRINTER'); END;
   12845   150         2          'L': BEGIN (*LIST ACTIVE HOSTS AND DEVICES*)
   12850   151         3                 WRITE(HELPINFO,'THE HOSTS AND DEVICES THAT ARE ACTIVE');
   12860   152         3                 WRITELN(HELPINFO,' ON THE NETWORK ARE AS FOLLOWS:');
   12870   153         3                 WRITELN(HELPINFO,'NETWORK CONFIGURATION TABLE PRINTED OUT'); END;
   12880   154         2          'S': BEGIN (*DESCRIBE SESSION CONTROL COMMANDS*)
   12890   155         3                 WRITE(HELPINFO,'THE SESSION CONTROL COMMANDS ARE');
   12900   156         3                 WRITELN(HELPINFO,' AS FOLLOWS:');
   12910   157         3                 WRITELN(HELPINFO,'NETWORK,LOGON,LOCALHOSTNAME');
   12920   158         3                 WRITELN(HELPINFO,'NETWORK,LOGOUT');
   12925   159         3                 WRITELN(HELPINFO);
   12930   160         3                 WRITE(HELPINFO,'THE LOCALHOSTNAME IS THE NAME OF THE');
   12940   161         3                 WRITELN(HELPINFO,' HOST TO WHICH ALL LOCAL COMMANDS WILL');
   12950   162         3                 WRITE(HELPINFO,'BE ROUTED AND NEED NOT BE THE NAME OF');
   12960   163         3                 WRITELN(HELPINFO,' THE HOST TO WHICH THE USER IS ');
   12970   164         3                 WRITE(HELPINFO,'PHYSICALLY CONNECTED.  IF NO LOCALHOSTNAME');
   12980   165         3                 WRITELN(HELPINFO,' IS SPECIFIED THEN THE HOST TO ');
   12990   166         3                 WRITE(HELPINFO,'WHICH THE USER IS ');
   13000   167         3                 WRITELN(HELPINFO,'PHYSICALLY CONNECTED IS ASSUMED'); END;
   13010   168         2        FILLER: BEGIN (*PROVIDE GENERAL INFORMATION*)
   13020   169         3                 WRITE(HELPINFO,'THE GENERAL COMMAND FORMATS ARE AS');
   13030   170         3                 WRITELN(HELPINFO,' FOLLOWS:');
   13040   171         3                 WRITELN(HELPINFO,'NETWORK,TRANSFER FILE,FILE TRANSFER PARAMETERS');
   13050   172         3                 WRITELN(HELPINFO,'NETWORK,LOGON,LOCALHOSTNAME');
   13060   173         3                 WRITELN(HELPINFO,'NETWORK,LOGOUT');
```

```
TESTEXTRNETCMD   4-DEC-1980   10:17:35   VAX-11 PASCAL VERSION V1.0-1        PAGE   5
01               4-DEC-1980   10:14:07   DMA1:[HOBART]EXTRNET.PAS;44 (4)

      LINE       LEVEL
    NUMBERS    PROC STMT STATEMENT.         .        .        .        .

  13070   174        3             WRITELN(HELPINFO,'NETWORK,HELP,HELPPARAM');
  13080   175        3             WRITELN(HELPINFO);
  13090   176        3             WRITELN(HELPINFO,'THE HELP REQUEST PARAMETERS ARE AS FOLLOWS:');
  13100   177        3             WRITELN(HELPINFO,'F)ILE TRANSFER');
  13110   178        3             WRITELN(HELPINFO,'L)IST CONFIGURATION');
  13120   179        3             WRITELN(HELPINFO,'S)ESSION COMMANDS');
  13130   180        3             WRITE(HELPINFO,'THE PARENTHESES IN THE ABOVE FORMATS');
  13140   181        3             WRITELN(HELPINFO,' INDICATE THAT ONLY THE FIRST LETTER ');
  13150   182        3             WRITELN(HELPINFO,'IS REQUIRED'); END
  13160   183        3        END; (*CASE*)
  13170   184        1
  13180   185        1
  13700   186        1   END; (*HELP USER*)
  13800   187        0
```

```
TESTEXTRNETCMD   4-DEC-1980   10:17:35   VAX-11 PASCAL VERSION V1.0-1        PAGE   6
01               4-DEC-1980   10:14:07   DMA1:[HOBART]EXTRNET.PAS;44 (5)

      LINE        LEVEL
     NUMBERS    PROC STMT STATEMENT.     .     .     .     .     .     .     .

             188         0
     13900   189    3          PROCEDURE CONTROLSESSION(CMDFIELD: PARTYPE;SESSPARAM: CMDTYPE;
     14000   190    3                                    LISTACTHOST: SETTYPE;VAR SESSIONMSG: TEXT);
     14100   191    3
     14200   192    3            CONST
     14300   193    3              MAXLOGMSGLNG=40;
     14400   194    3
     14500   195    3            VAR
     14600   196    3              I: INTEGER;
     14700   197    3              LOGMSG: PACKED ARRAY [1..MAXLOGMSGLNG] OF CHAR;
     14800   198    3              ERROR:  ERRORTYPE;
     14900   199    3              LOCALHOST: PARTYPE;
     14950   200    3              VALCOPY: BOOLEAN;
     15000   201    3
     15100   202    3            BEGIN (*CONTROL SESSION*)
     15300   203         0         IF (CMDFIELD[5]<>'N') AND (CMDFIELD[5]<>'U') THEN BEGIN
     15400   204         2           ERROR:='INVALID SESSION CONTROL COMMAND--CMD?>  ';
     15500   205         2           REPEAT GETPARAM(ERROR,CMDFIELD)
     15600   206         3             UNTIL (CMDFIELD[5]='N') OR (CMDFIELD[5]='U');
     15700   207         2           END; (*IF*)
     15800   208         1         IF CMDFIELD[5]='N' THEN BEGIN (*LOGON*)
     15825   209         2           I:=1;
     15850   210         2           COPY1PARAM(SESSPARAM,LOCALHOST,I,VALCOPY);
     15875   211         2           IF NOT VALCOPY THEN LOCALHOST[1]:='*';
     15900   212         2           IF NOT (LOCALHOST[1] IN LISTACTHOST) AND (LOCALHOST[1]<>FILLER)
     16000   213         2             THEN BEGIN   (*STUB FOR ERROR HANDLER*)
     16100   214         3             ERROR:='HOST NOT ACTIVE ON NETWORK--HOST NAME?> ';
     16200   215         3             REPEAT GETPARAM(ERROR,LOCALHOST)
     16300   216         4               UNTIL (LOCALHOST[1] IN LISTACTHOST) OR (LOCALHOST[1]=FILLER);
     16400   217         3             END; (*IF*)
     16500   218         2           IF LOCALHOST[1]<>FILLER THEN BEGIN
     16600   219         3             (*STUB FOR UPDATING COMMAND ROUTING TABLE*)
     16700   220         3             WRITE('COMMAND ROUTING TABLE UPDATED TO SHOW LOCAL HOST AS ');
     16800   221         3             CASE LOCALHOST[1] OF
     16900   222         4               'I': WRITELN('INTEL');
     17000   223         4               'N': WRITELN('NOVA');
     17100   224         4               'V': WRITELN('VAX')
     17200   225         4               END; (*CASE*)
     17300   226         3             REWRITE(SESSIONMSG);
     17400   227         3             WRITE(SESSIONMSG,'USER LOGGED ON NETWORK WITH ');
     17500   228         3             FOR I:=1 TO MAXPARLNG DO
     17550   229         3               IF LOCALHOST[I]<>FILLER THEN WRITE(SESSIONMSG,LOCALHOST[I]);
     17600   230         3             WRITELN(SESSIONMSG,' AS THE LOCAL HOST'); END (*THEN*)
     17610   231         3           ELSE BEGIN
     17615   232         3             REWRITE(SESSIONMSG);
     17620   233         3             WRITELN(SESSIONMSG,'USER LOGGED ON NETWORK WITH USER HOST AS LOCAL HOST'); END; (*IF*)
     17700   234         2           END (*THEN*)
     17800   235         2         ELSE IF CMDFIELD[5]='U' THEN BEGIN (*LOGOUT*)
     17900   236         2           (*STUB FOR RESETTING COMMAND ROUTING TABLE*)
     18000   237         2           WRITELN('COMMAND ROUTING TABLE SET BACK TO USER HOST');
     18100   238         2           REWRITE(SESSIONMSG);
     18200   239         2           (*STUB FOR FILE TRANSFER SUMMARY*)
     18300   240         2           WRITELN(SESSIONMSG,'SUMMARY OF FILE TRANSFERS');
     18400   241         2           WRITELN(SESSIONMSG,'USER LOGGED OUT'); END; (*IF*)
```

```
TESTEXTRNETCMD   4-DEC-1980   10:17:35   VAX-11 PASCAL VERSION V1.0-1        PAGE  7
01               4-DEC-1980   10:14:07   DMA1:[HOBART]EXTRNET.PAS;44 (5)

     LINE        LEVEL
    NUMBERS   PROC STMT STATEMENT.        .         .         .         .         .

  18500   242        1      (*END IF*)
  18600   243        1    END; (*CONTROL SESSION*)
  18700   244        0
```

```
TESTEXTRNETCMD   4-DEC-1980   10:17:35   VAX-11 PASCAL VERSION V1.0-1        PAGE   8
01               4-DEC-1980   10:14:07   DMA1:[HOBART]EXTRNET.PAS;44 (6)

      LINE         LEVEL
     NUMBERS    PROC STMT STATEMENT.         .         .         .         .         .         .

                     0
                3          PROCEDURE TRANSFERFILE(FILEPARAMS: CMDTYPE;
                3                           LISTACTDEV,LISTACTHOST: SETTYPE;VAR FTMSG: TEXT);
                3
                3            CONST
                3              ERRMSGLNG=40;
                3              MAXPARLNG=20;
                3
                3            TYPE
                3              PARTYPE=ARRAY [1..MAXPARLNG] OF CHAR;
                3              FILEPARAMSET=(DD,DH,FN,SD,SH);
                3
                3            VAR
                3              I: INTEGER;
                3              LEGSOURCEDEV,FPARAMSET: SET OF CHAR;
                3              FPARAM: FILEPARAMSET;
                3              DESTDEV,DESTHOST,FILENAME,INVPARAM,SOURCEDEV,SOURCEHOST: PARTYPE;
                3              VALPARAM,ERRORFLAG: BOOLEAN;
                3              PARAMSPRES: SET OF FILEPARAMSET;
                3              ERROR: PACKED ARRAY[1..ERRMSGLNG] OF CHAR;
                3
                3
                4          PROCEDURE SENDGETFILECMD(FILENAME,SOURCEDEV,SOURCEHOST,DESTDEV,
                4                                       DESTHOST: PARTYPE);
                4            BEGIN (*SEND GET FILE COMMAND*)
                     0         WRITE('FROM VAX TO ');
                     1         FOR I:=1 TO MAXPARLNG DO WRITE(SOURCEHOST[I]);
                     1         WRITELN;
                     1         WRITE('INFO FIELD=');
                     1         FOR I:=1 TO MAXPARLNG DO WRITE(FILENAME[I]);
                     1         FOR I:=1 TO MAXPARLNG DO WRITE(SOURCEDEV[I]);
                     1         FOR I:=1 TO MAXPARLNG DO WRITE(SOURCEHOST[I]);
                     1         FOR I:=1 TO MAXPARLNG DO WRITE(DESTDEV[I]);
                     1         FOR I:=1 TO MAXPARLNG DO WRITE(DESTHOST[I]);
                     1         WRITELN;
                     1       END; (*SEND GET FILE COMMAND*)
                     0
                     0
                     0       BEGIN (*TRANSFER FILE*)
                     0         PARAMSPRES:=[];
                     1         FPARAMSET:=['D','F','S'];
                     1         LEGSOURCEDEV:=LISTACTDEV-['C','P'];
                     1         I:=1;
                     1         WHILE FILEPARAMS[I]<>ENDLN DO BEGIN
                     2           IF I<>1 THEN I:=I+1;
                     2           IF FILEPARAMS[I] IN FPARAMSET THEN
                     2             CASE FILEPARAMS[I] OF
                     3               'D':  IF (FILEPARAMS[I+1]='D') AND (FILEPARAMS[I+2]='=')
                     3                         THEN BEGIN
                     4                           I:=I+3;
                     4                           COPY1PARAM(FILEPARAMS,DESTDEV,I,VALPARAM);
                     4                           IF VALPARAM AND (DESTDEV[1] IN LISTACTDEV) THEN
                     4                             PARAMSPRES:=PARAMSPRES+[DD]; END (*THEN*)
                     4                      ELSE IF (FILEPARAMS[I+1]='H') AND (FILEPARAMS[I+2]='=')
```

```
33900   TESTEXTRNETCMD   4-DEC-1980   10:17:35   VAX-11 PASCAL VERSION V1.0-1        PAGE   9
34000   01               4-DEC-1980   10:14:07   DMA1:[HOBART]EXTRNET.PAS;44 (6)

34100        LINE        LEVEL
34200       NUMBERS    PROC STMT  STATEMENT.         .        .        .        .        .        .
34300
34400     24100    299       3                       THEN BEGIN
34500     24200    300       4                         I:=I+3;
34600     24300    301       4                         COPY1PARAM(FILEPARAMS,DESTHOST,I,VALPARAM);
34700     24400    302       4                         IF VALPARAM AND (DESTHOST[1] IN LISTACTHOST) THEN
34800     24500    303       4                            PARAMSPRES:=PARAMSPRES+[DH]; (*END IF*)
34900     24600    304       4                         END (*THEN*)
35000     24700    305       4                     ELSE COPY1PARAM(FILEPARAMS,INVPARAM,I,VALPARAM);
35100     24800    306       3                     (*END IF*)
35200     24900    307       3            'F':  IF (FILEPARAMS[I+1]='N') AND (FILEPARAMS[I+2]='=')
35300     25000    308       3                       THEN BEGIN
35400     25100    309       4                         I:=I+3;
35500     25200    310       4                         COPY1PARAM(FILEPARAMS,FILENAME,I,VALPARAM);
35600     25300    311       4                         IF VALPARAM AND (FILENAME[1]<>FILLER) THEN
35700     25400    312       4                           PARAMSPRES:=PARAMSPRES+[FN]; END (*THEN*)
35800     25500    313       4                     ELSE COPY1PARAM(FILEPARAMS,INVPARAM,I,VALPARAM);
35900     25600    314       3                     (*END IF*)
36000     25700    315       3            'S':  IF (FILEPARAMS[I+1]='D') AND (FILEPARAMS[I+2]='=')
36100     25800    316       3                       THEN BEGIN
36200     25900    317       4                         I:=I+3;
36300     26000    318       4                         COPY1PARAM(FILEPARAMS,SOURCEDEV,I,VALPARAM);
36400     26100    319       4                         IF VALPARAM AND (SOURCEDEV[1] IN LISTACTDEV) THEN
36500     26200    320       4                           PARAMSPRES:=PARAMSPRES+[SD]; END (*THEN*)
36600     26300    321       4                     ELSE IF (FILEPARAMS[I+1]='H') AND (FILEPARAMS[I+2]='=')
36700     26400    322       3                       THEN BEGIN
36800     26500    323       4                         I:=I+3;
36900     26600    324       4                         COPY1PARAM(FILEPARAMS,SOURCEHOST,I,VALPARAM);
37000     26700    325       4                         IF VALPARAM AND (SOURCEHOST[1] IN LISTACTHOST) THEN
37100     26800    326       4                           PARAMSPRES:=PARAMSPRES+[SH]; END (*THEN*)
37200     26900    327       4                     ELSE COPY1PARAM(FILEPARAMS,INVPARAM,I,VALPARAM);
37300     27000    328       3                     (*END IF*)  END (*CASE*)
37400     27100    329       3           ELSE COPY1PARAM(FILEPARAMS,INVPARAM,I,VALPARAM); (*END IF*)
37500     27200    330       2           END; (*WHILE*)
37600     27300    331       1         ERRORFLAG:=FALSE;
37700     27400    332       1         FOR FPARAM:=DD TO SH DO
37800     27500    333       1           IF NOT (FPARAM IN PARAMSPRES) THEN BEGIN
37900     27600    334       2             ERRORFLAG:=TRUE;
38000     27700    335       2             CASE FPARAM OF
38100     27800    336       3               DD:  BEGIN (*DEST DEVICE*)
38200     27900    337       4                      ERROR:='INVALID OR MISSING DEST DEVICE--NAME?>  ';
38300     28000    338       4                      REPEAT GETPARAM(ERROR,DESTDEV)
38400     28100    339       5                        UNTIL DESTDEV[1] IN LISTACTDEV;
38500     28200    340       4                      END; (*DEST DEVICE*)
38600     28300    341       3               DH:  BEGIN (*DEST HOST*)
38700     28400    342       4                      ERROR:='INVALID OR MISSING DEST HOST--NAME?>    ';
38800     28500    343       4                      REPEAT GETPARAM(ERROR,DESTHOST)
38900     28600    344       5                        UNTIL DESTHOST[1] IN LISTACTHOST;
39000     28700    345       4                      END; (*DEST HOST*)
39100     28800    346       3               FN:  BEGIN (*FILE NAME*)
39200     28900    347       4                      ERROR:='INVALID OR MISSING FILE NAME--NAME?>    ';
39300     29000    348       4                      REPEAT GETPARAM(ERROR,FILENAME)
39400     29100    349       5                        UNTIL FILENAME[1]<>FILLER;
39500     29200    350       4                      END; (*FILE NAME*)
39600     29300    351       3               SD:  BEGIN (*SOURCE DEVICE*)
39700     29400    352       4                      ERROR:='INVALID OR MISSING SOURCE DEVICE--NAME?>';
```

```
TESTEXTRNETCMD   4-DEC-1980   10:17:35   VAX-11 PASCAL VERSION V1.0-1        PAGE  10
01               4-DEC-1980   10:14:07   DMA1:[HOBART]EXTRNET.PAS;44 (6)

      LINE        LEVEL
     NUMBERS    PROC STMT STATEMENT.           .         .         .         .         .

 29500   353       4                     REPEAT GETPARAM(ERROR,SOURCEDEV)
 29600   354       5                       UNTIL SOURCEDEV[1] IN LEGSOURCEDEV;
 29700   355       4                     END; (*SOURCE DEVICE*)
 29800   356       3              SH:  BEGIN (*SOURCE HOST*)
 29900   357       4                     ERROR:='INVALID OR MISSING SOURCE HOST--NAME?>  ';
 30000   358       4                     REPEAT GETPARAM(ERROR,SOURCEHOST)
 30100   359       5                       UNTIL SOURCEHOST[1] IN LISTACTHOST;
 30200   360       4                     END; (*SOURCE HOST*)
 30300   361       3              END; (*CASE*) END; (*IF*) (*END FOR*)
 30400   362       1    SENDGETFILECMD(FILENAME,SOURCEDEV,SOURCEHOST,DESTDEV,DESTHOST);
 30500   363       1    REWRITE(FTMSG);
 30510   364       1    FOR I:=1 TO MAXPARLNG DO
 30515   365       1      IF FILENAME[I]<>FILLER THEN WRITE(FTMSG,FILENAME[I]);
 30520   366       1    WRITE(FTMSG,' TRANSFERRED FROM ');
 30525   367       1    FOR I:=1 TO MAXPARLNG DO
 30530   368       1      IF SOURCEDEV[I]<>FILLER THEN WRITE(FTMSG,SOURCEDEV[I]);
 30535   369       1    WRITE(FTMSG,' ON ');
 30540   370       1    FOR I:=1 TO MAXPARLNG DO
 30545   371       1      IF SOURCEHOST[I]<>FILLER THEN WRITE(FTMSG,SOURCEHOST[I]);
 30550   372       1    WRITE(FTMSG,' TO ');
 30555   373       1    FOR I:=1 TO MAXPARLNG DO
 30560   374       1      IF DESTDEV[I]<>FILLER THEN WRITE(FTMSG,DESTDEV[I]);
 30565   375       1    WRITE(FTMSG,' ON ');
 30570   376       1    FOR I:=1 TO MAXPARLNG DO
 30575   377       1      IF DESTHOST[I]<>FILLER THEN WRITE(FTMSG,DESTHOST[I]);
 30580   378       1    WRITELN(FTMSG);
 30900   379       1  END; (*TRANSFER FILE*)
 31000   380       0
```

```
43100  TESTEXTRNETCMD   4-DEC-1980   10:17:35   VAX-11 PASCAL VERSION V1.0-1        PAGE  11
43200  01               4-DEC-1980   10:14:07   DMA1:[HOBART]EXTRNET.PAS;44 (7)

43300      LINE        LEVEL
43400     NUMBERS    PROC STMT STATEMENT.        .        .        .        .        .        .
43500
43600           381        0
43700   31100   382   3       PROCEDURE COPYALLPARAMS(SOURCESTRG: CMDTYPE;VAR PARAMSTRG: CMDTYPE;
43800   31200   383   3                                   INDEX: INTEGER);
43900   31300   384   3
44000   31400   385   3         VAR
44100   31500   386   3           J: INTEGER;
44200   31600   387   3
44300   31700   388   3         BEGIN (*COPY ALL PARAMETERS*)
44400   31800   389        0      J:=1;
44500   31900   390        1      WHILE (INDEX<MAXCMDLNG) AND (SOURCESTRG[INDEX]<>ENDLN) DO BEGIN
44600   32000   391        2        INDEX:=INDEX+1;
44700   32100   392        2        PARAMSTRG[J]:=SOURCESTRG[INDEX];
44800   32200   393        2        J:=J+1; END; (*WHILE*)
44900   32300   394        1      IF (SOURCESTRG[INDEX]<>ENDLN) OR (J=1) THEN PARAMSTRG[J]:=ENDLN;
45000   32400   395        1      WHILE J<MAXPARLNG DO BEGIN
45100   32500   396        2        J:=J+1;
45200   32600   397        2        PARAMSTRG[J]:=FILLER; END; (*WHILE*)
45300   32700   398        1    END; (*COPY ALL PARAMETERS*)
45400   32800   399        0
45500   32900   400        0
45600   33000   401        0
45700   33100   402        0
45800   33200   403        0
45900   33300   404        0    BEGIN (*EXECUTE TRANSMITTED NETWORK COMMAND*)
46000   33400   405        0      COMMANDSET:=['H','T','L'];
46100   33500   406        1      LISTACTDEV:=['C','F','H','P','T'];
46200   33600   407        1      LISTACTHOST:=['I','N','V'];
46300   33700   408        1      INDEX:=1;
46400   33800   409        1      COPY1PARAM(TRNETCMD,CMDFIELD,INDEX,VALIDCMD);
46500   34000   410        1      COPYALLPARAMS(TRNETCMD,PARAMS,INDEX);
46600   34100   411        1      REPEAT
46700   34200   412        2      ERRORMSG:=NONE;
46800   34300   413        2        IF VALIDCMD THEN
46900   34400   414        2          IF CMDFIELD[1] IN COMMANDSET THEN
47000   34500   415        2              CASE CMDFIELD[1] OF
47100   34600   416        3                'H': HELPUSER(PARAMS,NETRESP);
47200   34700   417        3                'L': CONTROLSESSION(CMDFIELD,PARAMS,LISTACTHOST,NETRESP);
47300   34800   418        3                'T': TRANSFERFILE(PARAMS,LISTACTDEV,LISTACTHOST,NETRESP);
47400   34900   419        3              END (*CASE*)
47500   35000   420        3          ELSE ERRORMSG:='INVALID COMMAND--COMMAND FIELD?>         '
47600   35100   421        2          (*END IF*) (*END THEN*)
47700   35200   422        2        ELSE ERRORMSG:='COMMAND FIELD TOO LONG--COMMAND FIELD?> ';
47800   35300   423        2        (*END IF*)
47900   35400   424        2        IF ERRORMSG<>NONE THEN GETPARAM(ERRORMSG,CMDFIELD);
48000   35500   425        2      VALIDCMD:=TRUE;
48100   35600   426        2      UNTIL ERRORMSG=NONE;
48200   35700   427        1    END; (*EXECUTE TRANSMITTED NETWORK COMMAND*)
48300   35800   428        0
48400   35900   429        0
```

```
TESTEXTRNETCMD   4-DEC-1980   10:17:35   VAX-11 PASCAL VERSION V1.0-1        PAGE  12
01               4-DEC-1980   10:14:07   DMA1:[HOBART]EXTRNET.PAS;44 (8)

       LINE        LEVEL
      NUMBERS   PROC STMT STATEMENT.          .          .          .          .          .

               430         0
      36000    431    2        PROCEDURE READINPUTSTRING;
      36100    432    2          BEGIN (*READ INPUT TEST STRING*)
      36200    433         0       INDEX:=0;
      36300    434         1       REPEAT
      36400    435         2         INDEX:=INDEX+1;
      36500    436         2         READ(SOURCE[INDEX]);
      36600    437         2       UNTIL SOURCE[INDEX]=ENDLN;
      36700    438         1       READLN;
      36800    439         1     END; (*READ INPUT TEST STRING*)
      36900    440         0
      37000    441         0
      37100    442    2        PROCEDURE PRINTTESTRESULTS;
      37200    443    2          VAR
      37300    444    2            CH: CHAR;
      37400    445    2
      37500    446    2          BEGIN (*PRINT TEST RESULTS*)
      37600    447         0       EXTRNETCMD(SOURCE,NETRESP);
      37700    448         1       RESET(NETRESP);
      37800    449         1       WHILE NOT EOF(NETRESP) DO BEGIN
      37900    450         2         WHILE NOT EOLN(NETRESP) DO BEGIN
      38000    451         3           READ(NETRESP,CH);
      38100    452         3           WRITE(CH); END; (*WHILE*)
      38200    453         2         READLN(NETRESP);
      38300    454         2         WRITELN; END; (*WHILE*)
      38400    455         1       WRITELN;  WRITELN;  WRITELN;
      38500    456         1     END; (*PRINT TEST RESULTS*)
      38600    457         0
      38700    458         0
      38800    459         0   BEGIN (*TEST EXECUTE TRANSMITTED NETWORK COMMAND*)
      38900    460         0     WRITELN('EXECUTING TEST PROGRAM--TYPE IN # TEST CASES');
      39000    461         1     READLN(NUMTESTCASES);
      39100    462         1     FOR I:=1 TO NUMTESTCASES DO BEGIN
      39200    463         2       WRITELN('TYPE IN TEST DESCRIPTION');
      39300    464         2       READLN;
      39400    465         2       WRITELN('TYPE IN TEST CASE #',I:0);
      39500    466         2       READINPUTSTRING;
      39600    467         2       PRINTTESTRESULTS; END; (*FOR*)
      39700    468         1   END. (*TEST EXECUTED TRANSMITTED NETWORK COMMAND*)
               469         0

Compilation time =    31.10 seconds (    905 lines/minute).

Active options at end of compilation:
NODEBUG,STANDARD,LIST,NOCHECK,WARNINGS,NOCROSS_REFERENCE,
NOMACHINE_CODE,OBJECT,ERROR_LIMIT = 30
```

Appendix F

Testing Documention

for the Network Command Language Interpreter

This appendix contains the final set of test cases that was used to test the network command language interpreter. Documentation on the rationale for each test input is included with the test case.

```
TYPE IN TEST DESCRIPTION
HELP REQUEST FOR FILE TRANSFER
TYPE IN TEST CASE #5
HELP,FILE TRANSFER!
FILE TRANSFER COMMAND FORMAT IS AS FOLLOWS:
NETWORK,T)RANSFER FILE,FN=FILENAME,DD=DESTDEV,DH=DESTHOST,SD=SOURCEDEV,SH=SOURCEHOST
T) MEANS THAT TRANSFER FILE MAY BE ABBREVIATED
PARAMETERS MAY BE ENTERED IN ANY ORDER
ALL PARAMETERS ARE LIMITED TO 20 CHAR
SOURCEDEV CANNOT BE THE CONSOLE OR PRINTER



TYPE IN TEST DESCRIPTION
HELP REQUEST FOR SESSION CONTROL COMMANDS
TYPE IN TEST CASE #6
HELP,SESSION COMMANDS!
THE SESSION CONTROL COMMANDS ARE AS FOLLOWS:
NETWORK,LOGON,LOCALHOSTNAME
NETWORK,LOGOUT

THE LOCALHOSTNAME IS THE NAME OF THE HOST TO WHICH ALL LOCAL COMMANDS WILL
BE ROUTED AND NEED NOT BE THE NAME OF THE HOST TO WHICH THE USER IS
PHYSICALLY CONNECTED.  IF NO LOCALHOSTNAME IS SPECIFIED THEN THE HOST TO
WHICH THE USER IS PHYSICALLY CONNECTED IS ASSUMED



TYPE IN TEST DESCRIPTION
HELP F\F\REQUEST FOR NETWORK CONFIGURATION
TYPE IN TEST CASE #7
HELP,LIST CONFIGURATION!
THE HOSTS AND DEVICES THAT ARE ACTIVE ON THE NETWORK ARE AS FOLLOWS:
NETWORK CONFIGURATION TABLE PRINTED OUT



TYPE IN TEST DESCRIPTION
ABBREVIATED GEN HELP REQUEST
TYPE IN TEST CASE #8
H!
THE GENERAL COMMAND FORMATS ARE AS FOLLOWS:
NETWORK,TRANSFER FILE,FILE TRANSFER PARAMETERS
NETWORK,LOGON,LOCALHOSTNAME
NETWORK,LOGOUT
NETWORK,HELP,HELPPARAM

THE HELP REQUEST PARAMETERS ARE AS FOLLOWS:
F)ILE TRANSFER
L)IST CONFIGURATION
S)ESSION COMMANDS
THE PARENTHESES IN THE ABOVE FORMATS INDICATE THAT ONLY THE FIRST LETTER
IS REQUIRED



TYPE IN TEST DESCRIPTION
ABBREVANAVIATED FILE TRANSFER HE,\,\LP REQUEST
TYPE IN TEST CASE #9
H,F!
FILE TRANSFER COMMAND FORMAT IS AS FOLLOWS:
NETWORK,T)RANSFER FILE,FN=FILENAME,DD=DESTDEV,DH=DESTHOST,SD=SOURCEDEV,SH=SOURCEHOST
```

```
EXECUTING TEST PROGRAM--TYPE IN # TEST CASES
20
TYPE IN TEST DESCRIPTION
invalid cmdfield
TYPE IN TEST CASE #1
[illegible]
INVALID COMMAND--COMMAND FIELD?>        logon
INVALID COMMAND--COMMAND FIELD?>        logon
INVALID COMMAND--COMMAND FIELD?>        llogon
INVALID COMMAND--COMMAND FIELD?>        help
INVALID COMMAND--COMMAND FIELD?>        1111111
INVALID COMMAND--COMMAND FIELD?>        hhhhhhhh
INVALID COMMAND--COMMAND FIELD?>        tttttttttt
INVALID COMMAND--COMMAND FIELD?>        LOGON
HOST NOT ACTIVE ON NETWORK--HOST NAME?> VAX
COMMAND ROUTING TABLE UPDATED TO SHOW LOCAL HOST AS VAX
USER LOGGED ON NETWORK WITH VAX AS THE LOCAL HOST


TYPE IN TEST DESCRIPTION
TYPICAL SESSION CONTROL COMMAND
TYPE IN TEST CASE #2
LOGOUT!
COMMAND ROUTING TABLE SET BACK TO USER HOST
SUMMARY OF FILE TRANSFERS
USER LOGGED OUT


TYPE IN TEST DESCRIPTION
TYPICAL FILE TRANSFER COMMAND
TYPE IN TEST CASE #3
TRANSFER FILE, FN-THESIS,DD SPINWRITER,DH=ZILOG,SD=HARDDISK,SH=VAX!
INVALID OF MISSING DEST DEVICE--NAME?>  INTEL
INVALID OR MISSING DEST DEVICE--NAME?>  PRINTER
INVALID OR MISSING IEST HOST--NAME?>    INTEL
INVALID OR MISSING FILE NAME--NAME?>    THESIS
FROM VAX TO VAX#################
INFO FIELD=THESIS###############HARDDISK############VAX##################PRINTER##############INTEL################
THESIS TRANSFERRED FROM HARDDISK ON VAX TO PRINTER ON INTEL


TYPE IN TEST DESCRIPTION
GENERAL HELP REQUEST
TYPE IN TEST CASE #4
HELP!
THE GENERAL COMMAND FORMATS ARE AS FOLLOWS:
NETWORK,TRANSFER FILE,FILE TRANSFER PARAMETERS
NETWORK,LOGON,LOCALHOSTNAME
NETWORK,LOGOUT
NETWORK,HELP,HELPPARAM

THE HELP REQUEST PARAMETERS ARE AS FOLLOWS:
F)ILE TRANSFER
L)IST CONFIGURATION
S)ESSION COMMANDS
THE PARENTHESES IN THE ABOVE FORMATS INDICATE THAT ONLY THE FIRST LETTER
IS REQUIRED
```

```
T) MEANS THAT TRANSFER FILE MAY BE ABBREVIATED
PARAMETERS MAY BE ENTERED IN ANY ORDER
ALL PARAMETERS ARE LIMITED TO 20 CHAR
SOURCEDEV CANNOT BE THE CONSOLE OR PRINTER


TYPE IN TEST DESCRIPTION
ABBREVIATED SESSION COMMANDS HELP REQWUEST
TYPE IN TEST CASE #10
H,S!
THE SESSION CONTROL COMMANDS ARE AS FOLLOWS:
NETWORK,LOGON,LOCALHOSTNAME
NETWORK,LOGOUT

THE LOCALHOSTNAME IS THE NAME OF THE HOST TO WHICH ALL LOCAL COMMANDS WILL
BE ROUTED AND NEED NOT BE THE NAME OF THE HOST TO WHICH THE USER IS
PHYSICALLY CONNECTED.  IF NO LOCALHOSTNAME IS SPECIFIED THEN THE HOST TO
WHICH THE USER IS PHYSICALLY CONNECTED IS ASSUMED


TYPE IN TEST DESCRIPTION
ABBREVIATED CONFIGURATION REQUEST
TYPE IN TEST CASE #11
H,L!
THE HOSTS AND DEVICES THAT ARE ACTIVE ON THE NETWORK ARE AS FOLLOWS:
NETWORK CONFIGURATION TABLE PRINTED OUT


TYPE IN TEST DESCRIPTION
NULL PARAMETER IN HELP REQUEST
TYPE IN TEST CASE #12
H,!
THE GENERAL COMMAND FORMATS ARE AS FOLLOWS:
NETWORK,TRANSFER FILE,FILE TRANSFER PARAMETERS
NETWORK,LOGON,LOCALHOSTNAME
NETWORK,LOGOUT
NETWORK,HELP,HELPPARAM

THE HELP REQUEST PARAMETERS ARE AS FOLLOWS:
F)ILE TRANSFER
L)IST CONFIGURATION
S)ESSION COMMANDS
THE PARENTHESES IN THE ABOVE FORMATS INDICATE THAT ONLY THE FIRST LETTER
IS REQUIRED


TYPE IN TEST DESCRIPTION
INVALID PARAMETER IN HELP REQUEST
TYPE IN TEST CASE #13
HELP,NETWORK!
INVALID HELP REQUEST PARAMETER--PARAM?>
THE GENERAL COMMAND FORMATS ARE AS FOLLOWS:
NETWORK,TRANSFER FILE,FILE TRANSFER PARAMETERS
NETWORK,LOGON,LOCALHOSTNAME
NETWORK,LOGOUT
NETWORK,HELP,HELPPARAM

THE HELP REQUEST PARAMFTERS ARE AS FOLLOWS:
F)ILE TRANSFER
```

```
L)IST CONFIGURATION
S)ESSION COMMANDS
THE PARENTHESES IN THE ABOVE FORMATS INDICATE THAT ONLY THE FIRST LETTER
IS REQUIRED


TYPE IN TEST DESCRIPTION
MULTIPLE PARAMETERS IN HELP REQUEST!
TYPE IN TEST CASE #14
HELP,L,S,F!
THE HOSTS AND DEVICES THAT ARE ACTIVE ON THE NETWORK ARE AS FOLLOWS:
NETWORK CONFIGURATION TABLE PRINTED OUT


TYPE IN TEST DESCRIPTION
SESSION CONTROL TEST OF INVALID LOCAL HOST
TYPE IN TEST CASE #15
LOGON,ZILOG!
HOST NOT ACTIVE ON NETWORK--HOST NAME?>
USER LOGGED ON NETWORK UITH USER HOST AS LOCAL HOST


TYPE IN TEST DESCRIPTION
SESSION CONTROL COMMAND WITH INVALID COMMAND
TYPE IN TEST CASE #16
LOGOFF
!
INVALID SESSION CONTROL COMMAND--CMD?>  LOGOUYT
COMMAND ROUTING TABLE SET BACK TO USER HOST
SUMMARY OF FILE TRANSFERS
USER LOGGED OUT


TYPE IN TEST DESCRIPTION
SESSION CONTROL COMMAND WITH NO LOCAL HOST SPECIFIED
TYPE IN TEST CASE #17
LOGON!
USER LOGGED ON NETWORK WITH USER HOST AS LOCAL HOST


TYPE IN TEST DESCRIPTION
SESSION CONTROL COMMAND SHOWING LOGIN ALLOWED
TYPE IN TEST CASE #18
LOGIN,VAX!
COMMAND ROUTING TABLE UPDATED TO SHOW LOCAL HOST AS VAX
USER LOGGED ON NETWORK WITH VAX AS THE LOCAL HOST


TYPE IN TEST DESCRIPTION
LOGOUT COMMAND WITH PARAMETERS
TYPE IN TEST CASE #19
LOGOUT,VAX!
COMMAND ROUTING TABLE SET BACK TO USER HOST
SUMMARY OF FILE TRANSFERS
USER LOGGED OUT
```

```
TYPE IN TEST DESCRIPTION
LOGON COMMAND WITH MULTIPLE LOCAL HOSTS SPECIFIED
TYPE IN TEST CASE #20
LOGON,VAX,INTEL,NOVA!
COMMAND ROUTING TABLE UPDATED TO SHOW LOCAL HOST AS VAX
USER LOGGED ON NETWORK WITH VAX AS THE LOCAL HOST



$ RUN EXTRNET
EXECUTING TEST PROGRAM--TYPE IN # TEST CASES
10
TYPE IN TEST DESCRIPTION
LOGON COMMAND FOR NOVA
TYPE IN TEST CASE #1
LOGON,NOVA!
COMMAND ROUTING TABLE UPDATED TO SHOW LOCAL HOST AS NOVA
USER LOGGED ON NETWORK WITH NOVA AS THE LOCAL HOST


TYPE IN TEST DESCRIPTION
LOGON COMMAND WITH MULTIPLE NULL PARAMETERS AND ONE VALID ONE
TYPE IN TEST CASE #2
LOGON,,,,,,,,,,,,,VAX!
USER LOGGED ON NETWORK WITH USER HOST AS LOCAL HOST


TYPE IN TEST DESCRIPTION
LOGON USING ABBREVIATIONS
TYPE IN TEST CASE #3
L,N!
INVALID SESSION CONTROL COMMAND--CMD?>  LOGON
COMMAND ROUTING TABLE UPDATED TO SHOW LOCAL HOST AS NOVA
USER LOGGED ON NETWORK WITH N AS THE LOCAL HOST


TYPE IN TEST DESCRIPTION
LOGON TO INTEL USING ABBREVIATION
TYPE IN TEST CASE #4
LOGON,I
!
COMMAND ROUTING TABLE UPDATED TO SHOW LOCAL HOST AS INTEL
USER LOGGED ON NETWORK WITH I  AS THE LOCAL HOST


TYPE IN TEST DESCRIPTION
HELP REQUEST WITH PARAMETER THAT IS TOO LONG
TYPE IN TEST CASE #5
HELP,FFFFFFFFFFFFFFFFFFFFFFFFFFF!
INVALID HELP REQUEST PARAMETER--PARAM?> FFFFFFFFFFFFFFFFFFFF
FILE TRANSFER COMMAND FORMAT IS AS FOLLOWS:
NETWORK,T)RANSFER FILE,FN=FILENAME,DD=DESTDEV,DH=DESTHOST,SD=SOURCEDEV,SH=SOURCEHOST
T) MEANS THAT TRANSFER FILE MAY BE ABBREVIATED
PARAMETERS MAY BE ENTERED IN ANY ORDER
ALL PARAMETERS ARE LIMITED TO 20 CHAR
SOURCEDEV CANNOT BE THE CONSOLE OR PRINTER
```

```
TYPE IN TEST DESCRIPTION
HELP REQUEST EWNWENWITH PARAMETER THAT IS EXACTLY 20 CHAR
TYPE IN TEST CASE #6
HELP,FFFFFFFFFFFFFFFFFFFF!
FILE TRANSFER COMMAND FORMAT IS AS FOLLOWS:
NETWORK,T)RANSFER FILE,FN=FILENAME,DD=DESTDEV,DH=DESTHOST,SD=SOURCEDEV,SH=SOURCEHOST
T) MEANS THAT TRANSFER FILE MAY BE ABBREVIATED
PARAMETERS MAY BE ENTERED IN ANY ORDER
ALL PARAMETERS ARE LIMITED TO 20 CHAR
SOURCEDEV CANNOT BE THE CONSOLE OR PRINTER


TYPE IN TEST DESCRIPTION
HELP REQUEST WEITH A PARAMETER THAT IS 21 CHARACTERS LONG
TYPE IN TEST CASE #7
H,LLLLLLLLLLLLLLLLLLLLL!
INVALID HELP REQUEST PARAMETER--PARAM?> LLLLLLLLL
THE HOSTS AND DEVICES THAT ARE ACTIVE ON THE NETWORK ARE AS FOLLOWS:
NETWORK CONFIGURATION TABLE PRINTED OUT


TYPE IN TEST DESCRIPTION
SESSION CONTROL COMMAND WITH A 20 CHAR PARAMETER
TYPE IN TEST CASE #8
LOGON,VVVVVVVVVVVVVVVVVVVV!
COMMAND ROUTING TABLE UPDATED TO SHOW LOCAL HOST AS VAX
USER LOGGED ON NETWORK WITH VVVVVVVVVVVVVVVVVVVV AS THE LOCAL HOST


TYPE IN TEST DESCRIPTION
SESSION CONTROL COMMAND WITH A 21 CHAR PARAMETER
TYPE IN TEST CASE #9
LOGON,IIIIIIIIIIIIIIIIIIIII!
HOST NOT ACTIVE ON NETWORK--HOST NAME?> IIIIIIIII
COMMAND ROUTING TABLE UPDATED TO SHOW LOCAL HOST AS INTEL
USER LOGGED ON NETWORK WITH IIIIIIIII AS THE LOCAL HOST


TYPE IN TEST DESCRIPTION
NULL COMMAND
TYPE IN TEST CASE #10
!
INVALID COMMAND--COMMAND FIELD?>         LOGOUT
COMMAND ROUTING TABLE SET BACK TO USER HOST
SUMMARY OF FILE TRANSFERS
USER LOGGED OUT
```

Appendix G

DELNET User's Manual

This appendix contains a user's manual for the network command language interpreter in its present version as documented by Appendix E. The manual consists basically of a description of the step-by-step procedure for using this program.

Since the lower-level protocols have not been implemented, it is necessary to log in to the VAX-11/780 to access the network command language interpreter. This log in procedure entails the following:

```
Username: HOBART <CR>
Password: XXX <CR>
WELCOME TO VAX VERSION 1.4
$
```

At this point, the network command language interpreter can be activated by typing NETWORK <CR> after the dollar sign prompt. The network command language interpreter will respond with

```
EXECUTING TEST PROGRAM--TYPE IN # TEST CASES
```

After an integer has been entered by the user, the network command language interpreter will respond

```
TYPE IN TEST DESCRIPTION
```

This allows the user to document what testing is being accomplished by this test input. A carriage return (<CR>) signals the end of the user input. The network command language interpreter will then respond

```
TYPE IN TEST CASE #1
```

Any of the commands described in the following sections may then be entered. The command must be terminated with an exclamation mark and a carriage return. The exclamation mark is used as a special character to represent the end of the network command.

File Transfers

A file transfer command consists of the following string:

```
TRANSFER FILE,FN=FILENAME,SD=SOURCE_DEVICE_NAME,
 SH=SOURCE_HOST_NAME,DD=DESTINATION_DEVICE_NAME,
 DH=DESTINATION_HOST_NAME!
```

The parameters may be entered in any order but may not be longer than 20 characters each. The maximum parameter length may be changed by changing the value of MAXPARLNG in the source code and recompiling and relinking the program.

Session Control

The network command language interpreter will accept the following session control commands:

```
LOGIN,HOST_NAME!
LOGOUT!
```

User Help Information

The user can request information by typing any of the following commands:

```
HELP!
HELP,FILE TRANSFER!
HELP,SESSION COMMANDS!
HELP,LIST CONFIGURATION!
```

The output from these help requests is shown on the following pages. Finally, Table 10 in the main body is included here also to show the valid keywords. The underlined letters are the allowable abbreviations.

```
[illegible]

[illegible] IS [illegible] TEST CASES

[illegible] DESCRIPTION
[illegible]
[illegible] TEST CASE #1

[illegible] COMMAND FORMATS ARE AS FOLLOWS.
[illegible] TRANSFER FILE, FILE TRANSFER PARAMETERS
[illegible] NAME
[illegible]
[illegible] PARAM

[illegible] PARAMETERS ARE AS FOLLOWS:
[illegible]
[illegible] CONFIGURATION
[illegible] COMMANDS
[illegible] IN THE ABOVE FORMATS INDICATE THAT ONLY THE FIRST LETT[illegible]
IS REQUIRED

[illegible] TEST DESCRIPTION
[illegible]
[illegible] TEST CASE #2

[illegible] TRANSFER COMMAND FORMAT IS AS FOLLOWS:
[illegible] TRANSFER FILE,FN=FILENAME,DD=DESTDE[illegible]SD=SOURCEDEV,SH=SOURCEHOST
[illegible] THAT TRANSFER FILE MAY BE ABBREVIATED
PARAMETERS MAY BE ENTERED IN ANY ORDER
ALL PARAMETERS ARE LIMITED [illegible]
SOURCEDEV CANNOT BE THE CONSOLE OR PRINTER

[illegible] TEST DESCRIPTION
[illegible]
[illegible]

[illegible] DEVICES THAT ARE ACTIVE ON THE NETWORK ARE AS FOLLOWS:
[illegible] CONFIGURATION TABLE PRINTED OUT

[illegible] DESCRIPTION
[illegible]
[illegible] TEST CASE [illegible]

[illegible] CONTROL COMMANDS ARE AS FOLLOWS:
[illegible] HOSTNAME
[illegible]

[illegible] IS THE NAME [illegible] WILL
```

BE ROUTED AND NEED NOT BE THE NAME OF THE HOST TO WHICH THE USER IS
PHYSICALLY CONNECTED. IF NO LOCALHOSTNAME IS SPECIFIED THEN THE HOST TO
WHICH THE USER IS PHYSICALLY CONNECTED IS ASSUMED

Table 10

Valid Keywords

Network Command Parameter

LOGIN
LOGOUT
TRANSFER FILE
HELP

File Transfer Parameters

Source Host (SH=), Destination Host (DH=),

INTEL
NOVA
VAX

Source Device (SD=)

FLOPPYDISK
HARDDISK
TAPE

Destination Device (DD=)

CONSOLE
FLOPPYDISK
HARDDISK
PRINTER
TAPE

Local Host Parameter

Null (Defaults to user host)
INTEL
NOVA
VAX

Help Parameter

FILE TRANSFER
LIST CONFIGURATION
SESSION COMMANDS

Vita

Captain William C. Hobart, Jr. was born on June 26, 1954 in Valpariso, Florida. In 1972, he graduated from Cabrillo Senior High School in Lompoc, California. He attended the United States Air Force Academy from which he received a Bachelor of Science degree with a major in Mathematics in 1976. Following graduation, he attended Communications Maintenance Officer School at Keesler AFB, Mississippi. Between April 1977 and June 1979, he was assigned to the Third Combat Communications Group, Air Force Communications Service at Tinker AFB, Oklahoma, as the maintenance officer for the Tactical Air Force Base 403 communications element. He entered the Air Force Institute of Technology in June 1979.

Permanent address: 778 Valley Green Dr
Brentwood, CA 94513

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GE/EE/81M-3 — 2. GOVT ACCESSION NO. AD-A100 822	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DESIGN OF A LOCAL COMPUTER NETWORK FOR THE AIR FORCE INSTITUTE OF TECHNOLOGY DIGITAL ENGINEERING LABORATORY	5. TYPE OF REPORT & PERIOD COVERED MS Thesis
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) William C. Hobart, Jr., Captain, USAF	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, OH 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, OH 45433	12. REPORT DATE March 1981
	13. NUMBER OF PAGES 319
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report)
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17 Frederic C. Lynch
Frederic C. Lynch, Major, USAF
Director of Public Affairs

2[illegible] 1

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Local Computer Network
X.25 Protocol
Computer Interfaces

21 MAY 1981

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See reverse

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract

A local computer for the Air Force Institute of Technology Digital Engineering Laboratory was designed and the network command language interpreter modules were implemented. The requirements for this network were specified by interviewing nine faculty members associated with the Digital Engineering Laboratory and then translating their functional requirements into a detailed set of hardware and software system requirements. Structured Analysis was used to produce a structured specification for the applications, host-to-host, network, and link protocol requirements. Yourdon and Constantine's Transform Analysis and Transaction Analysis techniques were then used to develop a set of module structure charts for the software design. The network uses a loop topology for the nodes with a star of up to four hosts connected to each node. The nodes are implemented using a Universal Network Interface Device (UNID) developed at the Air Force Institute of Technology. Initially, the network will include an Intel Series II Microcomputer Development Station, a Digital Equipment Corporation VAX-11/780, and a Data General Nova. These computers will be connected to the nodes using twisted pair and the two UNIDs in the initial configuration will be interconnected with a duplex fiber optic link supporting transmission rates up to 56 Kbs. The X.25 protocol was selected to implement a host-to-host transfer mechanism in conjunction with a basic routing algorithm using a lookup table stored in each UNID. The network command language interpreter allows file transfer commands, session control commands, and user help requests to be parsed and the appropriate parameters passed to lower-level modules.

ATE

LMED

8